Universidad Carlos III de Madrid

Escuela Politécnica Superior

PROYECTO FIN DE CARRERA

Ingeniería Técnica en Informática de Gestión

Desarrollo de un simulador de aprendizaje no supervisado.

Autor: Luis Losa Domínguez

Director: José María Valls Ferrán

Leganés, Octubre de 2015

Índice

Índice de ilustraciones

Índice de tablas

1. Introducción y objetivos.

1.1 Introducción

Las redes de neuronas son modelos matemáticos que se construyen a partir de ejemplos y permiten resolver gran cantidad de problemas reales. La construcción de modelos a partir de ejemplos se encuadra dentro de un área llamada Aprendizaje Automático. Este aprendizaje puede ser supervisado, cuando con cada ejemplo le decimos a la red la salida que debería obtener, y por tanto la red puede saber el error cometido con cada ejemplo a la hora de construir el modelo. Otro tipo de aprendizaje es el no supervisado donde no existe una salida que la red debe obtener con cada ejemplo, es decir no hay una información del error que se comete con cada dato cuando la red se está entrenando. La red por sí sola debe extraer la información de los datos de entrada.

Una de los modelos de redes de neuronas más utilizados que realizan aprendizaje no supervisado, son los Mapas Autoorganizados de Kohonen.

La motivación principal de este proyecto es la realización de una aplicación que simule el funcionamiento de una red de neuronas no supervisada basada en los mapas autoorganizados de Kohonen. Esta aplicación se ha realizado como un proyecto de ingeniería de software completo, con todas sus fases de análisis, diseño, implementación y pruebas.

1.2 Objetivo

El objetivo principal de este proyecto es realizar una aplicación plenamente operativa que realice el entrenamiento de una red de neuronas basándose en los mapas autoorganizados de Kohonen.

Además, esta aplicación puede servir como una herramienta didáctica para los alumnos de ingeniería informática donde puedan comprender de una manera práctica los mapas de Kohonen.

La herramienta debe solicitar al alumno que interactúe con la aplicación, siendo el alumno quien elija los datos y los parámetros que se utilizarán en los algoritmos. La aplicación ejecutará los algoritmos y mostrará al usuario los resultados de estos.

Otros objetivos parciales son:

- La aplicación debe tener un uso sencillo e intuitivo

- La aplicación debe ser robusta: en caso de error debe informar al usuario de ello sin dejar de funcionar.
- Aunque los mapas de Kohonen realizan un aprendizaje no supervisado, a veces hay salidas disponibles en los datos y éstos están etiquetados. En este caso puede aprovecharse esta situación para etiquetar las neuronas correspondientes, lo que se denomina "calibrado" y clasificar nuevos datos. Esta importante característica se contemplará en esta aplicación.
- Se realizará una aplicación web para que el alumno no tenga que instalar ni descargar nada y pueda realizar la práctica de una forma rápida desde un interfaz fácil e intuitivo.

1.3 Estructura de la memoria

A continuación explicaremos como está estructurada la memoria, haciendo una breve explicación de las partes más importantes.

La memoria consta de los siguientes capítulos:

- Estado del arte: La función de este capítulo es poner en situación al lector con unas nociones teóricas acerca de los temas más importantes que se están tratando en la memoria.

- Contenido de la aplicación: Este capítulo nos muestra el contenido de la aplicación con el fin de que el lector se haga una idea de las distintas partes que forman la aplicación. Se hará un repaso de las distintas pantallas que forman el aplicativo y se explicarán en detalle qué acciones puede realizar el usuario interactuando con el sistema.

- Diseño de la aplicación: En este capítulo se explicará cómo se ha diseñado la aplicación, contando cómo se ha diseñado la estructura de paquetes e indicando la función de cada uno de ellos. También se hace un repaso de las clases más importantes de cada uno de ellos, explicando la función, atributos y métodos más importantes de cada clase.

- Desarrollo del sistema: En este capítulo se explica cómo se ha desarrollado la aplicación, como se ha programado la aplicación, explicando los algoritmos más importantes.

- Conclusiones: Este capítulo cuenta las conclusiones que hemos sacado una vez terminada la aplicación y posibles líneas futuras.

1.4 Fases del desarrollo

La metodología de desarrollo utilizada en este proyecto es la metodología de desarrollo en cascada que ordena rigurosamente las etapas del desarrollo de software donde cada etapa debe esperar a que acabe la anterior. Las etapas que forman esta metodología son: toma de requisitos, análisis, diseño, implementación y pruebas.

Ilustración 1. Desarrollo en cascada

En una primera fase de requerimientos se realizaron distintas reuniones con José M. y analizamos qué requisitos íbamos a necesitar para realizar el proyecto. Las reuniones se repitieron varias veces las primeras semanas hasta que tuvimos claro que era lo que teníamos que hacer. En esta fase quedaron claros cuales eran los requisitos de nuestro proyecto.

Luego tuvimos la fase de análisis donde estuvimos analizando cómo íbamos a resolver los requerimientos tomados en la fase previa. En esta fase fue donde decidimos que realizaríamos una aplicación web además de analizar los algoritmos que íbamos a escribir para solucionar los distintos requisitos que teníamos definidos para realizar la aplicación y lo realizamos con frecuentes reuniones.

En la fase de diseño las reuniones fueron más espaciadas ya que tuve que realizar el diseño de la aplicación. Primero fue la arquitectura del proyecto web, creando un proyecto web java estructurando las distintas partes en que iba a constar el proyecto. También diseñe los estilos que iba a tener el proyecto creando una página tipo con el menú, la cabecera, contenido y pie de página. Se realizaron reuniones para enseñar a José M. como quedaría el diseño y cambios que hicimos de la primera versión.

La cuarta fase fue la de implementación donde programé las distintas funcionalidades que requería la aplicación. Esta fue la fase más larga donde mantuvimos varias reuniones viendo cómo íbamos creando las distintas partes de la aplicación, comprobando que se realizaba de acuerdo a lo que habíamos pensado anteriormente.

La última fase fue la de pruebas donde probamos que la aplicación funcionaba correctamente probando cada una de las funcionalidades que debía realizar la aplicación de acuerdo a los requisitos funcionales recogidos en la primera fase del proyecto. En esta última fase comprobamos en alguna reunión el funcionamiento y dimos paso a escribir la memoria del proyecto.

1.5 Presupuesto y planificación

Para calcular el presupuesto total del proyecto debemos incluir los costes del material necesario para realizarlo más los costes de personal.

Los costes materiales que debemos contabilizar en el proyecto incluyen los costes hardware y software. Los costes imputables se hacen en base a la siguiente formula:

$$\text{Coste imputable} = \frac{\text{Dedicación}}{\text{Período de depreciación}} \cdot \text{Uso dedicado} \cdot \text{Coste sin IVA}$$

Los costes hardware solo incluyen un ordenador portátil que se ha comprado para realizar el proyecto por 549 €. Las características del portátil son:

- Toshiba C550-C-10j
    - Procesador AMD A-8 7410 2.5 GHz. 4 nucleos
    - 16 GB de RAM
    - S.O. Windows 8

Los costes hardware imputables son:

CONCEPTO	DEDICACIÓN (MESES)	PERIODO DE (MESES) DEPRECIACIÓN	COSTE SIN IVA	COSTE IMPUTABLE
Toshiba C550-C-10j	3	48	549 €	34.30 €

Tabla 1. Costes hardware

El único software que hemos utilizado en el proyecto no gratuito ha sido Microsoft Office 2013. Los demás programas utilizados son gratuitos. Los costes software son los siguientes:

CONCEPTO	DEDICACIÓN (MESES)	PERIODO DE (MESES) DEPRECIACIÓN	COSTE SIN IVA	COSTE IMPUTABLE
Microsoft Office 2013	3	48	57,02 €	3,56 €

Tabla 2. Costes software

Para realizar el proyecto se necesita un equipo formado por un jefe de proyecto, un analista, un analista- programador y un programador.

En el siguiente diagrama de Gantt se pueden observar las distintas tareas que forman la planificación del proyecto y la persona del equipo que realizará la tarea. La fase de análisis la realizará conjuntamente el jefe de proyecto junto con el analista con el fin de analizar los distintos requisitos de la aplicación.

La fase de análisis la realizan el analista-programador y el programador donde conjuntamente realizarán la implementación del proyecto.

En la última tarea de pruebas todas las partes del equipo trabajarán conjuntamente para probar las distintas partes de la aplicación para comprobar el buen funcionamiento de la misma.

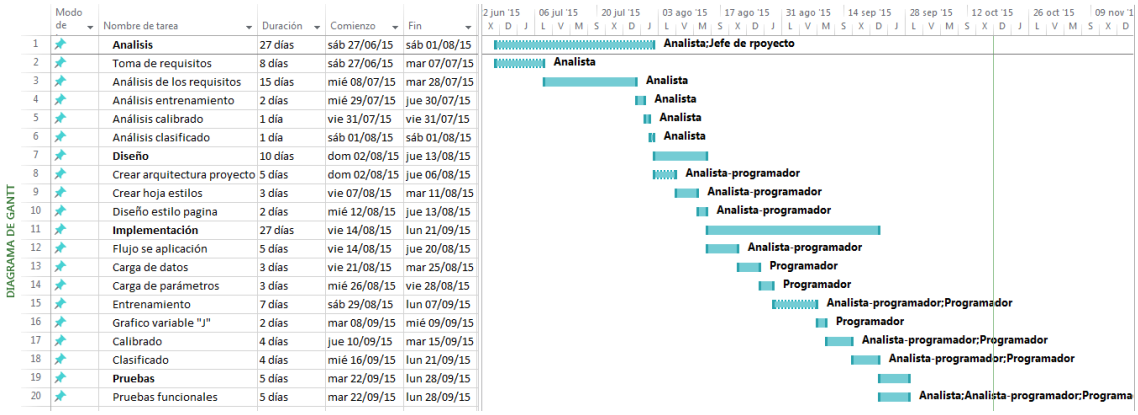

Ilustración 2. Diagrama de Gantt

Los costes del personal se pueden ver desglosados en la siguiente tabla en función de las horas trabajadas y el perfil que se desempeña:

EQUIPO	HORAS TRABAJADAS	COSTE HORA (€)	TOTAL
Jefe de proyecto	256	85	21.760 €
Analista	256	46	11.776 €
Analista-programador	352	39	13.728 €
Programador	272	33	8.976 €

Tabla 3. Costes de personal

El coste de realización del proyecto asciende a 68.096,21 euros. Los costes del proyecto se pueden ver desglosados en la siguiente tabla resumen:

Concepto	Coste
Hardware	34,30 €
Software	3,56 €
Personal	56.240 €
Total sin IVA	56.277,86 €
IVA (21%)	11.818,35 €
Total	68.096,21 €

Tabla 4. Coste del proyecto

2. Estado del arte

2.1 Redes de neuronas

Un campo importante dentro de la Inteligencia Artificial son las redes de neuronas, que se inspiran en el comportamiento del cerebro humano y crean modelos artificiales que tratan de solucionar problemas que son difíciles de resolver utilizando técnicas algorítmicas convencionales.

La aparición de los primeros computadores y el desarrollo de las primeras teorías del aprendizaje y procesamiento neuronal se produjeron en la década de los 40.

Una neurona se define como un dispositivo que tiene un conjunto de entradas y que genera una única salida. Y una red de neuronas está formada por un conjunto de neuronas interconectadas entre sí.

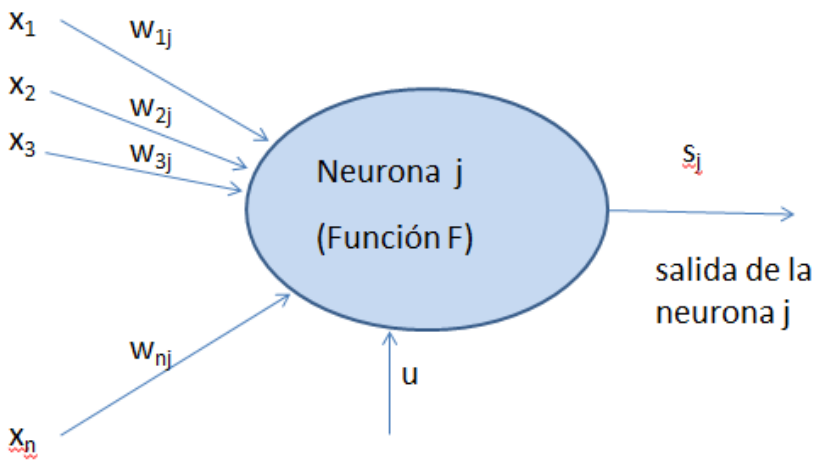

Ilustración 3. Cálculo de la salida de una neurona

La siguiente formula calcula la salida de una neurona:

$$sj = F\left(\sum x_i w_{ij} + u\right)$$

Las neuronas tienen un conjunto de entradas que pueden provenir de las salidas de otras neuronas o de los datos de entrada de la red. Cada conexión tiene asociada un peso, cuya representación será "Wij", siendo i y j las neuronas conectadas por ella.

Para realizar el cálculo de la salida de una neurona debemos sumar todas las entradas que recibe la neurona, multiplicadas por los pesos de sus conexiones más el umbral de la propia neurona. A este valor resultante de las entradas se le aplica una función de activación (representada como F). La salida de cada neurona se propaga desde las conexiones recibidas hacia el final de la red.

Una red de neuronas se puede ver como una función matemática que recibe unos datos de entrada y produce unos datos de salida.

El objetivo principal será generar un modelo matemático que sea capaz de ajustarse a los datos utilizados de la forma más exacta posible.

Para generar el modelo, debemos determinar la arquitectura de la red, es decir, el número de capas y neuronas que existen en la red y la conectividad que existirá entre ellas. También habrá que seleccionar las funciones de activación a emplear y la regla de aprendizaje. La red se inicializará con valores aleatorios de los pesos y utilizará como ejemplos los datos de entrenamiento.

Una vez entrenada la red obtendremos el conjunto de los valores finales de los pesos de las conexiones de la red que permiten que el modelo matemático se ajuste a los datos de entrenamiento.

Las redes más habituales utilizan aprendizaje supervisado, es decir cada dato o ejemplo contiene una salida deseada y por tanto es posible medir el error que se produce al comparar la salida deseada con la salida que produce la red. En función de este error se modificarán los pesos en el aprendizaje.

Consideraremos que la red habrá aprendido del conjunto de entrenamiento cuando el error producido para todo el conjunto de datos sea menor que un valor dado.

El método de ajuste consiste en ir introduciendo sucesivamente los datos de entrada y a continuación calcular la salida que produce la red y compararla con la salida deseada. Los pesos se irán ajustando en función de esa diferencia. Este ajuste se hace de manera que la próxima vez que a la red le sea presentado el mismo ejemplo, pueda producir una salida con menor error. Una práctica habitual es introducir varias veces el conjunto de ejemplos para permitir un ajuste mayor y gradual de los pesos de la red, ya que una vez que un ejemplo ha sido caracterizado, el tratar de aprender otro nuevo puede hacer "olvidar" el previamente aprendido.

Es importante que la red, una vez que finalice su entrenamiento, sea capaz de generalizar, es decir de responder adecuadamente ante datos que no se han utilizado en el aprendizaje. Para evaluar la capacidad de generalización de una red, habrá que

medir el error que comete la red utilizando datos que no se han utilizado durante el entrenamiento. El conjunto de estos datos se denomina conjunto de test.

2.2 Mapas autoorganizados de Kohonen (SOM)

Un mapa autoorganizado (SOM por sus siglas en inglés) es un tipo de red neuronal artificial que es entrenada usando aprendizaje no supervisado para producir una representación discreta del espacio llamado mapa.

En el aprendizaje no supervisado no existe la salida deseada, por lo que la red no puede guiarse por el error obtenido, ya que no es posible medir dicho error.

El modelo fue descrito por primera vez por Teuvo Kohonen debido a lo cual son llamados a veces mapas de Kohonen.

Los mapas de Kohonen son usados en la práctica para abordar diferentes problemas como pueden ser:

- agrupamiento o *clustering*
- minería de datos (*data mining)*
- análisis, diagnóstico, monitorización y control de procesos
- aplicaciones biomédicas, incluyendo métodos de diagnóstico y análisis de datos en bioinformática
- análisis de datos en comercio, industria, macroeconomía y finanzas.

Los mapas autoorganizados son un tipo de red de neuronas que tiene dos capas, la capa de entrada y la capa de competición o de salida, como se puede observar en la siguiente figura:

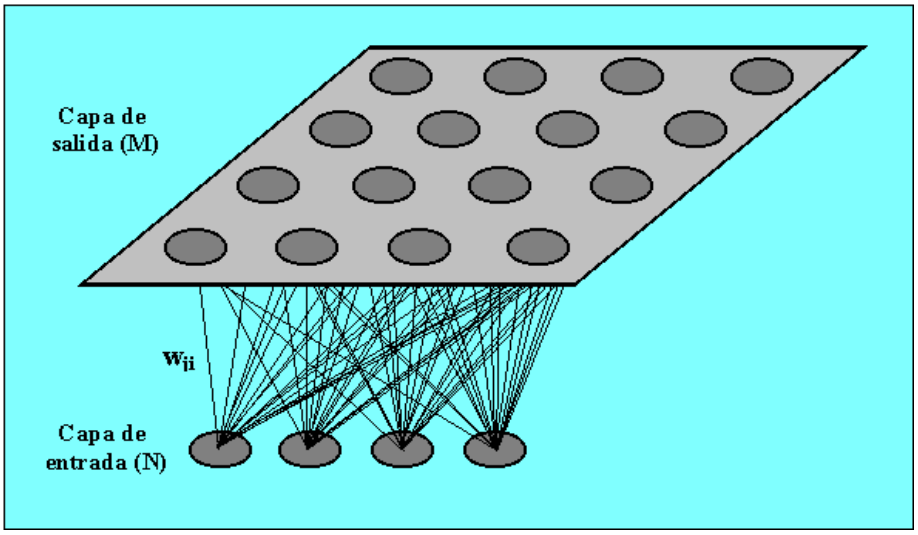

Ilustración 4. Mapa autoorganizados

La capa de entrada está formada por n neuronas, tantas como atributos tienen los datos de entrada y se encarga de recibir y transmitir a la capa de salida la información procedente del exterior. Cada una de las neuronas está conectada con todas las neuronas de la capa de salida por medio de unos pesos. Cada dato de entrada se transmite desde la capa de entrada a la capa de salida, y es en esta capa donde se calcula la activación de cada neurona para ese dato. Los datos de entrada deben ser vectores numéricos donde cada valor tiene que ser un número real.

La capa de salida o de competición se encarga de procesar la información y formar el mapa de características. Su estructura suele ser bidimensional. El número de células que forman la capa de salida se especifica en dos parámetros al crear la red que indicarán el número de filas y columnas que tendrá la capa, multiplicándolos nos dará el número de neuronas que forman la capa. A cada neurona le llega una señal de la capa de entrada y se calcula su activación, y es la que tiene mayor activación a la que llamamos célula ganadora. Se entenderá como mayor activación la de la neurona con la menor distancia euclídea al dato de entrada. La función de distancia euclídea la podemos ver en la ilustración. A esta neurona y a sus vecinas son a las que se actualizan los pesos en función del aprendizaje.

Para calcular las células vecinas se aplica la función de vecindario normalmente con un límite máximo de vecindad. Se pueden utilizar diferentes tipos de distancia de vecindario. El límite de vecindario es dinámico ya que va decreciendo en cada ciclo.

$$d_{ij} = \sqrt{\sum_{p=i}^{k} (x_{ip} - x_{jp})^2}$$

Ilustración 5. Función distancia euclídea

La configuración usual de las neuronas en el mapa de dos dimensiones puede tener topología rectangular o hexagonal como se observa en la siguiente figura:

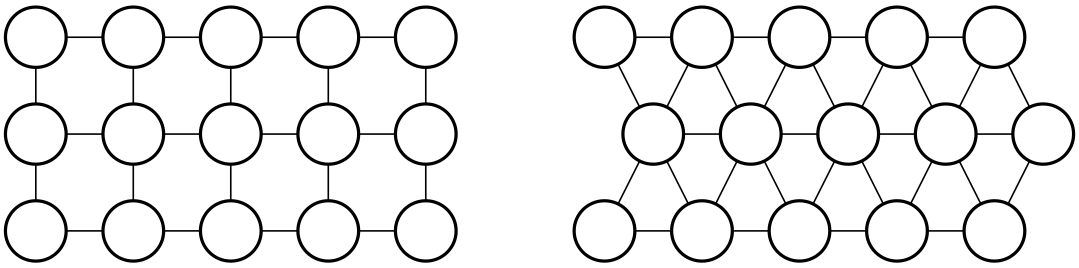

Ilustración 6. Topología rectangular y hexagonal

Otro parámetro importante es la tasa de aprendizaje, cuyo valor es un número real entre 0 y 1 que nos indica cómo variarán los pesos de las células al ser actualizadas cuando ha sido ganadora o vecina. La tasa de aprendizaje va decreciendo en cada ciclo del entrenamiento al igual que lo hace el límite de vecindario. Si ambos parámetros no comienzan con valores bajos implica que el mapa sufrirá grandes cambios en los primeros ciclos del entrenamiento y variaciones más moderadas en los últimos ciclos.

La ley de aprendizaje que se aplica sobre las células es la siguiente:

$$\Delta\mu_{ij} = \begin{cases} \dfrac{\alpha(t)}{\sigma(c_k, c_j)}\left(e_i(t) - \mu_{ij}(t)\right), \text{si } c_k \text{ es la ganadora y } \sigma(c_k, c_j) \leq \theta(t) \\ 0, \text{en caso contrario} \end{cases}$$

Ilustración 7. Ley de aprendizaje

donde:

- μ_{ij} es el peso i de la neurona j. Las neuronas tienen tantos pesos como atributos tienen los datos de entrada.
- $\alpha \in [0,1]$ es la tasa o razón de aprendizaje.
- $\sigma(c_k, c_j)$ es el grado de vecindad entre las neuronas i y j. Se define como: $\sigma(c_k, c_j) = 1 + d(c_k, c_j)$ siendo $d(c_i, c_j)$ la distancia entre las neuronas i y j según la función de distancia de vecindario elegida.
- e_i es el atributo i del patrón de entrada e.
- $\theta \in \mathbb{R}^+ \geq 1$ es el límite de vecindario.

El entrenamiento de un mapa de Kohonen podría sintetizarse en los siguientes pasos:

1. Inicializar los pesos con valores aleatorios pequeños.
2. Presentar un patrón de entrada.
3. Calcular la distancia del patrón presentado a cada neurona de la red.
4. Seleccionar la neurona ganadora, la más cercana al patrón presentado.
5. Obtener las vecinas de la ganadora y sus distancias a ella.
6. Aplicar la ley de aprendizaje.
7. Volver al paso 2, hasta haber presentado el conjunto completo de patrones de entrada.
8. Actualizar la tasa de aprendizaje (α) y el límite de vecindario (θ).
9. Si se han hecho menos ciclos de los indicados, volver al paso 2. En caso contrario, fin.

Como ya hemos comentado antes, este tipo de red de neuronas realiza aprendizaje no supervisado.

A veces se dispone de datos etiquetados, es decir con una salida, igual que ocurre en aprendizaje supervisado. Para que sea posible aprovechar esta información, una vez realizado el aprendizaje que siempre es no supervisado, podemos etiquetar las neuronas con la salida mayoritaria de los datos más próximos a ellas. A este proceso le llamamos "calibrado". Calibrar el mapa consiste en etiquetar las células. Para saber qué etiqueta corresponde a una neurona se realizará un cálculo de las etiquetas de los datos de los que la célula es su prototipo, es decir, cuando la neurona salió como ganadora. La etiqueta que más se repita en los datos del prototipo se asignará a la célula.

Recordamos que en el proceso de entrenamiento el método de aprendizaje es no supervisado y que la calibración se realiza una vez terminado el entrenamiento. Una vez terminado el etiquetado de las células es posible que alguna no tenga etiqueta ya que nunca ha salido como ganadora

Una vez calibrado el mapa podemos realizar la clasificación de los datos. La clasificación consiste en asignar a cada dato una neurona. Se asigna la célula más cercana y si esta célula tiene etiqueta se le asigna al dato también.

Los pasos que podrían darse para realizar la clasificación de los datos son los siguientes:

1. Configurar los parámetros del mapa y entrenarlo (este paso no tiene en cuenta que los datos están etiquetados).
2. Calibrar el mapa utilizando las clases de los datos.
3. Utilizando la calibración hecha en el paso 2, comprobar la tasa de aciertos de entrenamiento y de test.
4. Volver al paso 1 si se quieren probar distintos mapas.

2.3 Software existente de SOM: SOM_PAK

Uno de los programas software más destacados para utilizar este tipo de red de neuronas es "SOM_PAK", que fue desarrollado por el Centro de Investigación de Redes de Neuronas Artificiales del Laboratorio de Computación y Ciencias de la Información de la Universidad de Tecnología de Helsinki.

Este software es de dominio público y en vez de presentarse como una sola aplicación, está formado por varios programas que son ejecutados a través de la línea de comandos y que conjuntamente ofrecen toda la funcionalidad.

Estos programas son entre otros:

- randinit: la función de este programa es inicializar los vectores de pesos de forma aleatoria. Los parámetros de entrada de este programa son las dimensiones del mapa, un archivo de entrada, un archivo de salida, el tipo de vecindario y la topología del mapa.

- vsom: este programa sirve para entrenar el mapa. Será necesario introducir como entrada del programa la cantidad de ciclos, los valores iniciales de la tasa de aprendizaje y del límite de vecindario, además de un archivo de entrada, que es el de salida del programa randinit y otro de salida.

- vcal: este programa calibra el mapa en función de las clases especificadas para cada patrón de entrada.

- qerror: la función de este programa es cuantificar el error cometido por la red.

- sammon: genera una imagen (la proyección de Sammon) con el objetivo de poder ofrecer cierta información sobre los datos desde un punto de vista visual.

A continuación, vamos a exponer un ejemplo de uso sencillo, que incluye los pasos básicos. El programa ofrece más opciones de las especificadas en el ejemplo y para realizar un uso completo de las funcionalidades debe consultarse la documentación del mismo.

En el primer paso vamos a inicializar el mapa:

```
> randinit -din train.dat -cout map.cod -xdim 12 –ydim 8
  -topol hexa -neigh bubble –rand 123
```

Este comando en primer lugar inicializa el mapa para realizar el entrenamiento con los datos del archivo "train.dat" y genera como archivo de salida "map.cod". El mapa estará formado por 8 filas y 12 columnas ("-xdim" indica la dimensión en el eje X y "-ydim" la dimensión en el eje Y). El mapa creado tendrá topología hexagonal, el tipo de vecindario será "bubble" y los pesos se inicializarán aleatoriamente usando el número 123 como semilla.

El paso siguiente que se realiza es el entrenamiento:

```
> vsom -din train.dat -cin map.cod -cout map.cod –rlen 1000
  –alpha 0.05 -radius 10
```

El mapa será entrenado con los mapas del archivo "train.dat", usando el mapa inicializado anteriormente "map.cod" y se guarda en "map.cod". El entrenamiento constará de 1000 ciclos o iteraciones y comenzará con una tasa de aprendizaje de 0.05 y una distancia de límite de vecindario inicial de 10.

Una vez realizado el entrenamiento, podemos cuantificar el error cometido con el conjunto de datos y el mapa entrenado:

```
> qerror –din train.dat –cin map.cod
```

Además, si el conjunto de datos está etiquetado, es posible calibrar el mapa:

```
> vcal –din train.dat –cin map.cod –cout map.cod  -
```

2.4 Aplicaciones web

Llamamos aplicación web a aquellas herramientas que los usuarios pueden utilizar mediante un navegador web que se conecta a través de internet o de una intranet, a un servidor remoto.

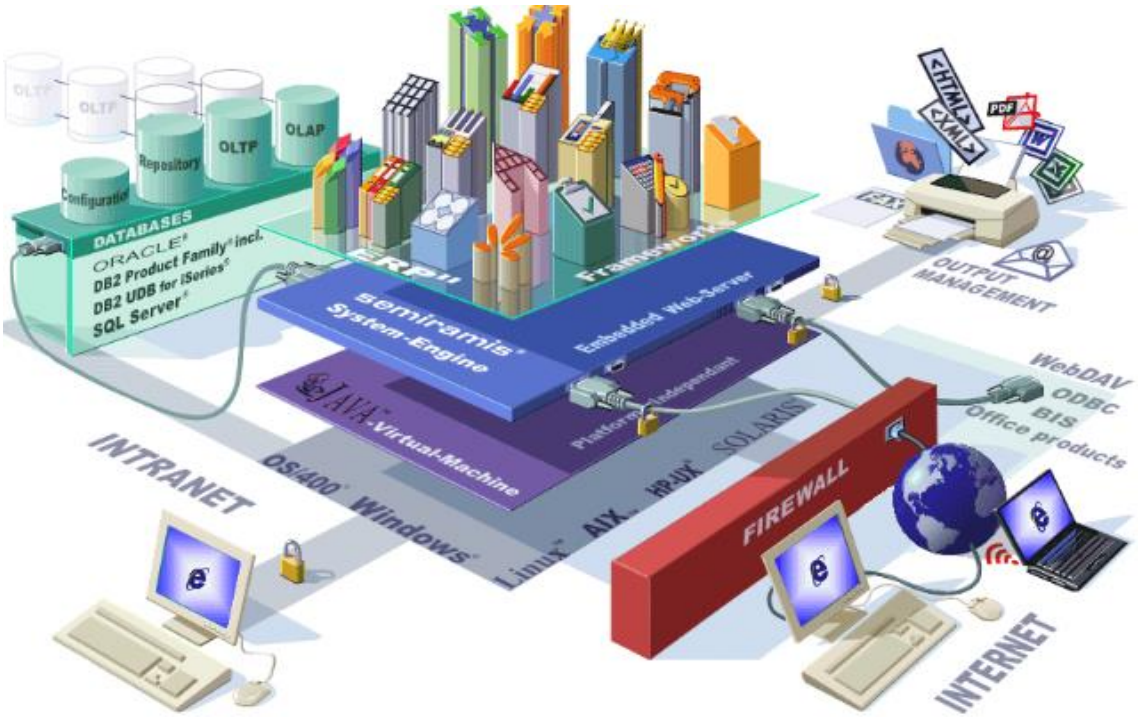

Ilustración 8. Aplicación web

La popularidad de las aplicaciones web se debe a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo y a la facilidad de mantener y actualizar aplicaciones web sin tener que hacerlo en miles de usuarios finales, ya que si queremos realizar una mejora o actualización en el programa hay que hacerlo una sola vez en el servidor donde se encuentra la aplicación.

Una aplicación web contiene elementos que proporcionan una comunicación activa entre el usuario y la información, permitiendo al usuario acceder a los datos de modo interactivo, recibiendo las respuestas de la aplicación web a las acciones del usuario, como por ejemplo rellenando formularios, participando en juegos o accediendo a bases de datos.

En los principios de la programación cliente-servidor, cada aplicación tenía como interfaz de usuario, su propio programa cliente, que tenía ser instalado en cada computadora de cada usuario. El interfaz realizaba peticiones a otro programa, el

servidor, que le daba las respuestas. Si había que realizar alguna mejora o actualización en el programa muchas veces había que hacerlo también en el interfaz, por lo que había que realizarlos en cada computadora de cada usuario lo que conlleva mayor coste y menos productividad.

Las aplicaciones web funcionan de otra manera, generan una serie de páginas en formato html, xhtml que son soportados por los navegadores web. Se utilizan lenguajes como javascript, java, php, etc en el lado cliente para añadir elementos dinámicos al interfaz. También se utilizan tecnologías como Ajax que permiten al desarrollador añadir más funcionalidades que ofrecen una experiencia interactiva sin tener que recargar la página.

Una de las ventajas más significativas de las aplicaciones web es que funcionan igual independientemente del sistema operativo instalado en el cliente por lo que la aplicación se escribe una sola vez y se ejecuta igual en todas partes.

La estructura de una aplicación web normalmente está compuesta de tres capas, un navegador, un motor y una base de datos. El navegador web realiza peticiones a la capa intermedia que se sirve de la base de datos mediante consultas y actualizaciones, para proporcionar una respuesta al usuario.

A continuación enumeramos algunas de las ventajas de las aplicaciones web:

- Ahorra tiempo, sin necesidad de descargas ni de instalar ningún programa.
- No hay problemas de compatibilidad.
- No ocupan espacio.
- Actualizaciones inmediatas.
- Consumo bajo de recursos.
- Multiplataforma.
- Portables.
- Disponibilidad alta
- Los virus no dañan.
- Colaboración.
- Los navegadores ofrecen cada vez más y mejores funcionalidades para crear aplicaciones.

Algunas desventajas de las aplicaciones web es que ofrecen menos funcionalidades que las aplicaciones de escritorio ya que las funcionalidades que ofrecen los sistemas operativos son mayores que las que ofrecen los navegadores. Y otra desventaja también es que la funcionalidad depende de un tercero que es el proveedor de la conexión a internet.

2.5 Java

Java es un lenguaje de programación de propósito general, orientado a objetos, diseñado para tener las menos dependencias posibles de implementación. La intención es que las aplicaciones se escriban una vez y se puedan ejecutar en cualquier dispositivo, (conocido en inglés como WORA, o "write once, run anywhere"), que quiere decir que el código puede ser ejecutado en cualquier dispositivo independientemente de su arquitectura y su sistema operativo mientras tenga instalada la máquina virtual de Java (JVM) y no tiene que volver a ser recompilado. Java es uno de los lenguajes de programación más usados actualmente, principalmente en aplicaciones web cliente-servidor.

Los orígenes de Java se remontan a 1990 cuando un equipo de Sun Microsystem dirigido por James Gosling comienza el desarrollo de software para dispositivos electrónicos. En 1993 con la aparición de Internet se percata de la potencialidad de su lenguaje para mejorar la interactividad de las páginas web en internet y decide implementarlo en esa dirección. En 1995 fue publicado como un componente fundamental de la plataforma Java de Sun Microsystems.

Los principales objetivos con los que se creó Java son:

- Debe ser un lenguaje de programación orientado a objetos.
- Un mismo programa se debe poder ejecutar en múltiples sistemas operativos.
- Debe incluir por defecto soporte para trabajo en red.
- Debe ser diseñado para ejecutar código en sistemas remotos de forma segura.
- Debe ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos

La orientación a objetos se basa en la idea de diseñar el software de forma que los distintos tipos de datos que se usen, estén unidos a sus operaciones. Para ello los datos y el código se combinan en entidades llamadas objetos. Normalmente los cambios en las estructuras de datos implican cambios en las operaciones que actúan sobre los datos y la programación mediante objetos ofrece una base más estable en el diseño de aplicaciones. Mediante la creación de objetos fomentamos la reutilización de software entre proyectos que es una de las premisas de la ingeniería software.

El lenguaje Java deriva de lenguajes como C++ pero a diferencia de este que combina la sintaxis para programación genérica, estructurada y orientada a objetos, Java está únicamente fue construido orientado a objeto. Prácticamente todo en Java es un objeto y cada objeto reside en una clase, que sería el molde a partir del cual se crean los objetos.

3. Contenido de la aplicación

En este capítulo vamos a explicar el contenido de la aplicación. Para ello iremos mostrando las distintas pantallas de la aplicación haciendo una explicación del contenido de las mismas así como la función que tienen las mismas.

Para realizar estas capturas de pantalla hemos utilizado un conjunto de datos muy conocido denominado "pima-indians-diabetes" que puede encontrarse en http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes.

3.1 Carga de datos

La primera pantalla de la aplicación es la de carga de datos. En ella se cargarán el fichero de entrenamiento y opcionalmente el fichero que contenga el mapa inicial.

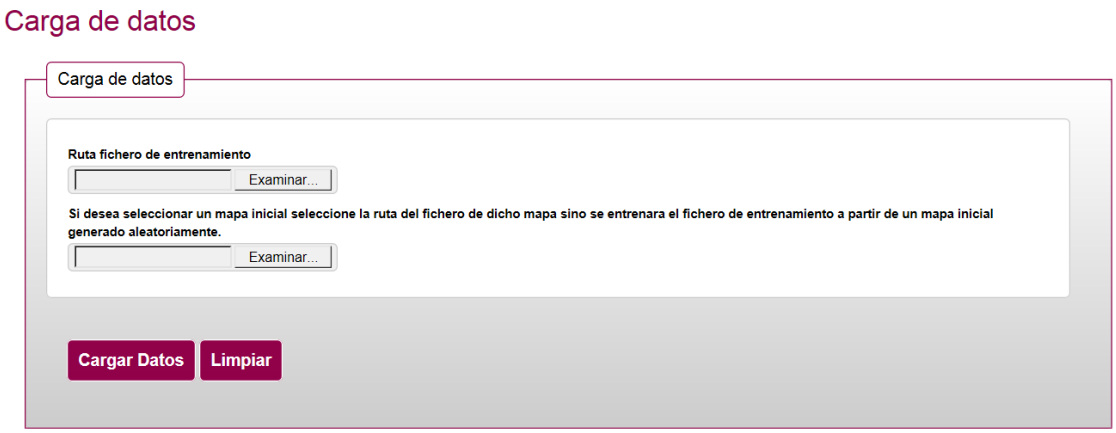

Ilustración 9. Pantalla de carga de datos

En esta pantalla se seleccionara la ruta del fichero de datos inicial o también llamado de entrenamiento.

Dicho fichero debe tener el formato correcto puesto que la aplicación lo validará. El fichero debe tener una primera línea con un número entero que especificará la dimensión de los datos del fichero. El resto de líneas contendrán los datos donde cada línea corresponderá a un dato con la dimensión especificada y cada dato puede tener sus valores separados por una coma, espacio o tabulador. Al final de cada línea cada dato puede tener una etiqueta.

Este fichero será utilizado para realizar el entrenamiento del mapa.

Opcionalmente se podrá seleccionar un fichero que contenga un mapa inicial. Este fichero también será validado y deberá tener el mismo formato que el fichero de entrenamiento. Aunque el fichero de entrenamiento se refiere a datos y el fichero con el mapa inicial a células o neuronas el formato es el mismo ya que ambos, datos y células, se representan mediante valores que representan un punto en el espacio n-dimensional, siendo n la dimensión, que es el valor que se encuentra en la primera línea. La dimensión de este mapa debe coincidir con la del fichero de entrenamiento ya que si no, la aplicación lo validará y devolverá el error cuando se realice el entrenamiento.

Una vez seleccionados los ficheros el usuario pinchará en el botón "Cargar Datos" y la aplicación cargará en memoria la ruta del fichero de entrenamiento seleccionado y del fichero con el mapa inicial si el usuario lo seleccionó y se pasará a la siguiente pantalla.

En nuestro ejemplo práctico hemos seleccionado el fichero diabetes.txt que contiene los datos que hemos comentado. Cada línea contiene un dato con ocho valores que coincide con la dimensión del valor de la primera línea que es un ocho.

3.2 Carga de parámetros

La siguiente pantalla de la aplicación es la de "Carga de parámetros". En esta pantalla se cargarán los parámetros que se aplicarán como entrada del algoritmo de entrenamiento.

Ilustración 10. Pantalla de carga de parámetros

Los parámetros son los siguientes:

- En primer lugar se indicarán las dimensiones “x” e “y” del mapa que nos darán el número de células que tendrá el mapa inicial en caso de que el usuario no seleccione un fichero que contenga un mapa inicial. Estos campos sólo aceptan números enteros. Si el usuario introduce un 4 en la dimensión “x” y un 6 en la dimensión “y” el mapa inicial contendrá 24 células creadas aleatoriamente.

- El siguiente parámetro es el número de ciclos, que deberá también ser un número entero ya que ese número definirá el número de veces que se va a ejecutar el algoritmo de entrenamiento.

- La tasa de aprendizaje inicial es otro parámetro que deberá poder cargar un número con decimales y se utilizará para el cálculo del aprendizaje de la célula ganadora y las células vecinas.

- El tipo de variación de la tasa de aprendizaje es el último parámetro de la pantalla. Este parámetro indicará el tipo de variación que se aplicará a la tasa de aprendizaje. El tipo de variación que se aplicará deberá ser un seleccionable con las siguientes opciones:

    - Variación lineal.
    - Variación logarítmica.
    - Sin variación.

- El límite de vecindario es el último parámetro y deberá ser un número entero y se utilizará para encontrar las células vecinas.

3.3 Operaciones

La siguiente pantalla de la aplicación es la de “Operaciones”. Esta pantalla contiene cuatro botones cuya función explicaremos a continuación:

- Entrenar: Mediante este botón se ejecuta el algoritmo de entrenamiento.

- Calibrar: Al pulsar este botón la aplicación nos redirige a la pantalla de calibrado.

- Clasificar: Pulsando este botón la aplicación nos redirige a la pantalla de clasificado.

- Volver: Este botón nos lleva de nuevo a la pantalla de “Carga de parámetros”.

Ilustración 11. Pantalla de operaciones

3.4 Entrenamiento

Si hemos pulsado el botón “Entrenar” la aplicación ejecutará el algoritmo de entrenamiento pasándole los parámetros recogidos en las pantallas de carga de datos y carga de parámetros.

Una vez terminado el algoritmo de entrenamiento la aplicación nos redirige a la pantalla de resultado de entrenamiento:

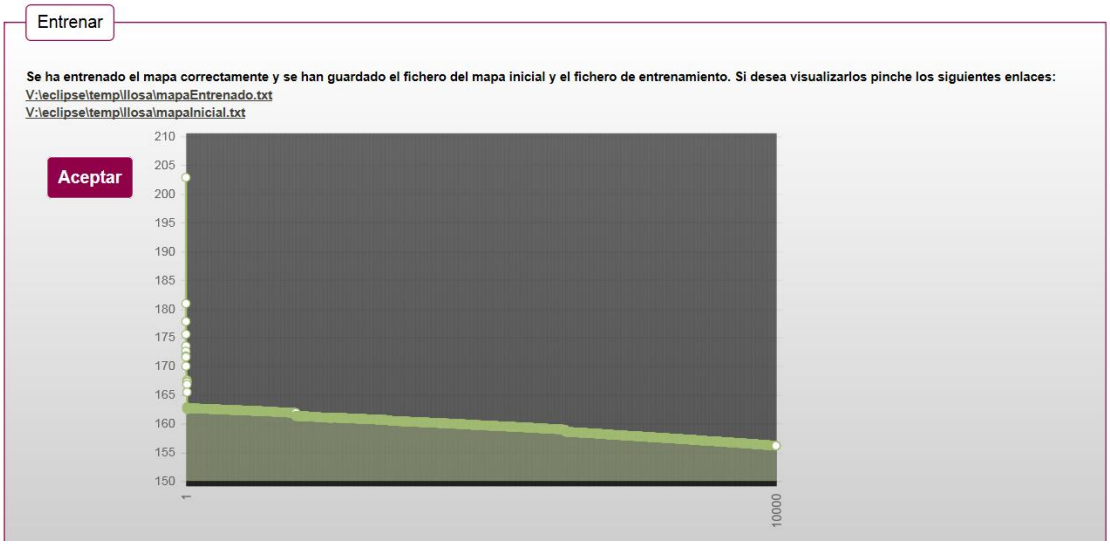

Ilustración 12- Pantalla con el resultado del entrenamiento

Esta pantalla nos muestra el resultado del entrenamiento. Si el algoritmo ha concluido de manera correcta la pantalla muestra un mensaje de que el algoritmo ha terminado correctamente. Si se ha producido algún tipo de error durante la ejecución del entrenamiento el sistema nos muestra por pantalla el correspondiente error.

En esta pantalla también aparecen dos enlaces. Ambos enlaces nos muestran las rutas donde la aplicación ha guardado los ficheros del mapa entrenado y del mapa inicial en el servidor.

El primer enlace nos abre un fichero de texto con el mapa entrenado. El segundo enlace nos abre el fichero que contiene el mapa inicial que se utilizó al principio del entrenamiento.

Si el usuario pincha en un enlace se abrirá automáticamente un fichero de texto con el mapa como muestra la siguiente imagen:

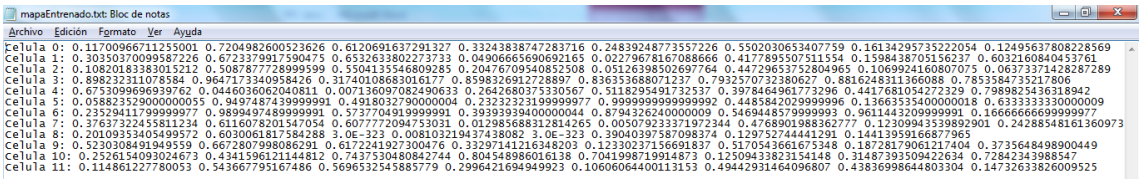

Ilustración 13. Fichero de mapa entrenado

Además esta pantalla nos muestra un gráfico con la evolución que ha ido teniendo la variable "J" en cada ciclo durante la ejecución del algoritmo de entrenamiento. Esta variable nos indica, como ya veremos en el siguiente capítulo, la suma de las distancias de cada punto a su célula ganadora en cada iteración. La suma de las distancias totales en cada iteración es lo que se muestra en el gráfico de la figura anterior.

Es en este gráfico donde podemos ver cómo evoluciona la suma de las distancias a medida que se ejecuta el algoritmo y en esta figura podemos observar como la variable va disminuyendo en cada ciclo y se puede determinar que el algoritmo funciona correctamente ya que la suma de las distancias cada vez es menor. Si la evolución de la variable se fuera incrementando o no disminuyera deberemos comprobar el algoritmo ya que no se estaría ejecutando correctamente

3.5 Calibrado

Si pulsamos el botón "Aceptar" en la pantalla de resultado del entrenamiento volveremos a la pantalla de operaciones. En la pantalla de operaciones si pulsamos el botón "Calibrar" la aplicación nos mostrará la pantalla de calibrado:

Ilustración 14. Pantalla de calibrado

Estamos en la pantalla de calibrado. En esta pantalla podemos realizar distintas acciones.

Si pulsamos el botón volver la aplicación nos redirigirá de nuevo a la pantalla de operaciones.

Para realizar el calibrado lo primero que tenemos que hacer es seleccionar el fichero de datos con el que queremos calibrar el mapa entrenado. Tenemos dos opciones, si dejamos en blanco el seleccionable el algoritmo utilizará para el calibrado del mapa el fichero de entrenamiento.

En caso de querer realizar el calibrado del mapa con otro fichero de datos lo seleccionaremos y pulsaremos el botón "Calibrar". Una vez pulsamos éste botón se ejecuta el algoritmo de calibrado y una vez terminado la aplicación nos redirige a la pantalla de resultado de calibrado:

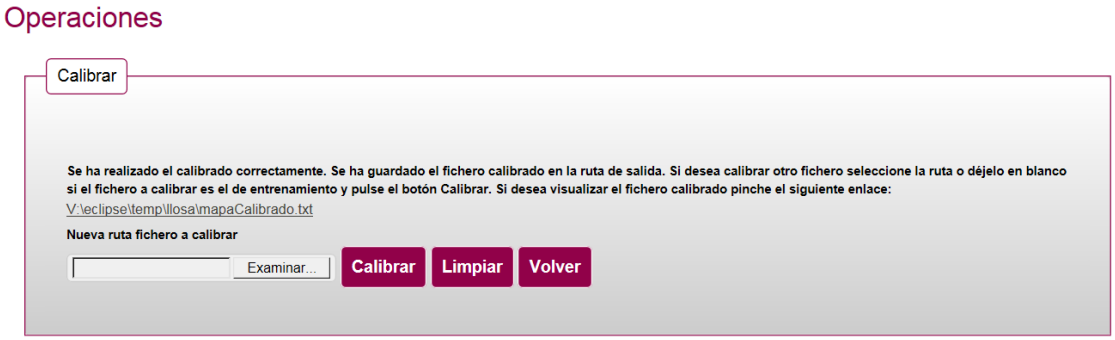

Ilustración 15. Pantalla de resultado de calibrado

En esta pantalla observamos cómo ha resultado el calibrado del mapa, si el algoritmo se ha ejecutado correctamente o si se ha producido algún tipo de error mediante el mensaje correspondiente que se muestra en la pantalla.

Esta pantalla también nos muestra un enlace al fichero de texto que contiene el mapa calibrado, que el sistema previamente ha guardado en el servidor. Si pinchamos en el enlace nos abrirá el fichero de texto con el mapa calibrado:

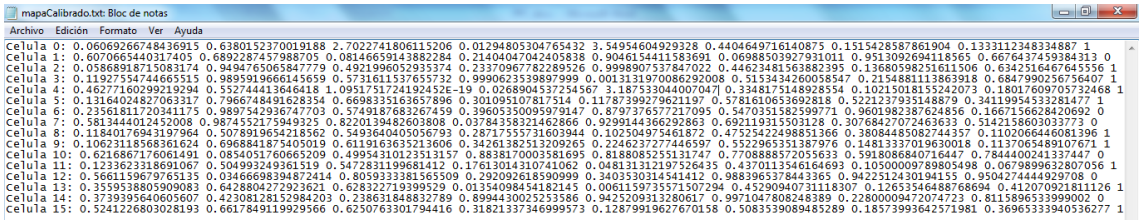

Ilustración 16. Fichero de texto con el mapa calibrado

En este fichero podemos ver el resultado del mapa calibrado.

El fichero que hemos usado en el ejemplo es el fichero diabetes.txt. Este fichero tiene dos etiquetas 0 y 1 que indican si el paciente es diabético (1) o no (0). Al realizar el algoritmo de calibrado se realiza el proceso de etiquetado de las células que componen el mapa y en el fichero podemos observar que a cada célula le ha puesto una etiqueta (0 y 1) que indica que esa célula tiene más datos de diabéticos o no diabéticos más cerca en el mapa. Por ejemplo la célula 0 tiene la etiqueta 1 y quiere decir que esa célula tiene más cercanos los datos de diabéticos. La célula 1 tiene por el contrario la etiqueta 0 que nos indica que reúne más datos de personas no diabéticas. De esta manera tenemos el mapa calibrado y ahora puede usarse para realizar el clasificado de algún fichero de datos.

En está pantalla de calibrado podemos volver a ejecutar el algoritmo las veces que deseemos, únicamente tendremos que seleccionar de nuevo el fichero a calibrar y pulsamos el botón “Calibrar“.

3.6 Clasificado

Si pulsamos el botón “Volver” la aplicación nos retornará a la pantalla de operaciones. Si pulsamos el botón “Clasificar” la aplicación nos llevará a la pantalla de clasificado:

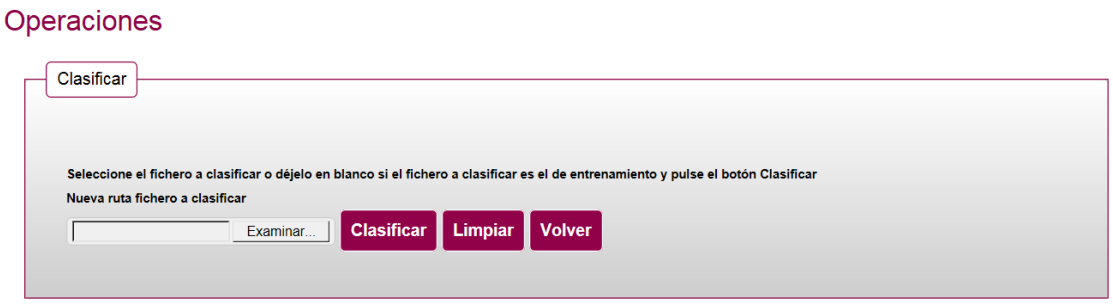

Ilustración 17. Pantalla de clasificado

Esta pantalla es la pantalla “Clasificar”. En esta pantalla vamos a poder clasificar un fichero de datos en función del mapa calibrado.

Al igual que en la pantalla de calibrado en está pantalla tenemos la opción de clasificar los datos de entrenamiento o podemos clasificar los datos de un fichero que elijamos por medio del seleccionable. Si lo dejamos en blanco clasificará los datos del fichero de entrenamiento. Una vez elijamos el fichero que queremos clasificar pincharemos en el botón “Clasificar” y se ejecutará el algoritmo de clasificado.

Una vez termine el algoritmo la aplicación nos redirigirá a la pantalla de resultado de clasificado:

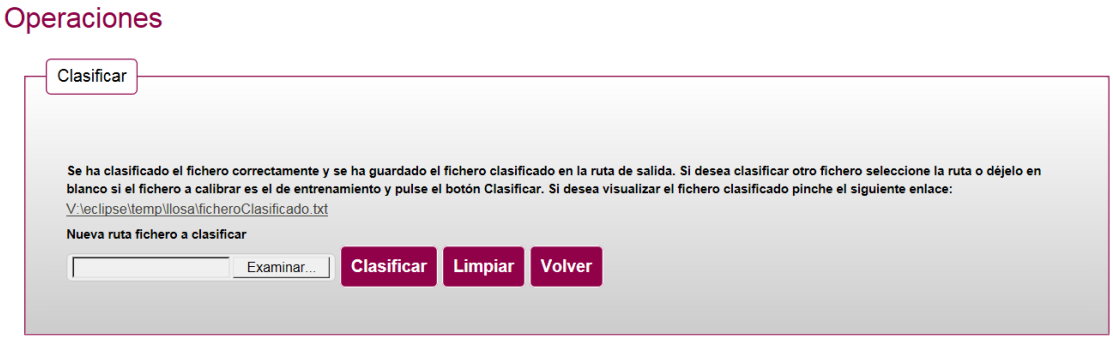

Ilustración 18. Pantalla de resultado de clasificado

En esta pantalla podemos observar cómo ha resultado el algoritmo de clasificado, si ha terminado de manera correcta o si se ha producido algún tipo de error mediante el mensaje que muestra la pantalla.

En la pantalla vemos cómo el algoritmo ha terminado correctamente.

La pantalla nos muestra un enlace con los datos del fichero clasificado después de ejecutarse el algoritmo de clasificado, que la aplicación ha guardado en el servidor de aplicaciones. Si pinchamos dicho enlace nos abrirá el fichero de texto con los datos clasificados:

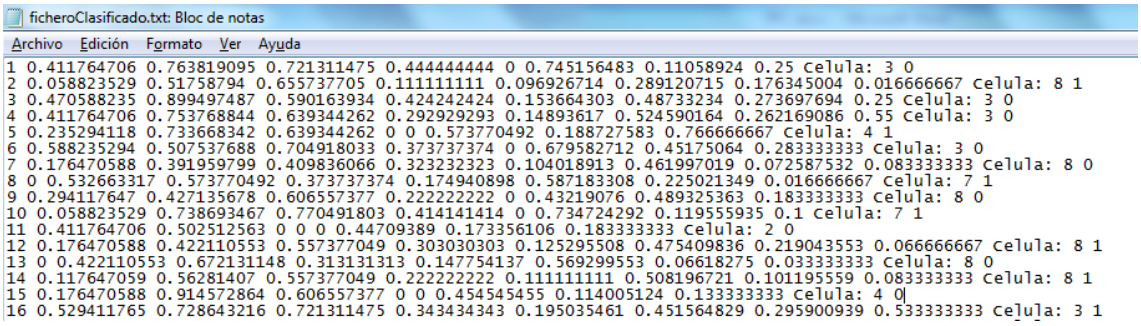

Ilustración 19. Fichero con los datos clasificados

Está imagen nos muestra parte del fichero de datos clasificados. En el fichero se puede observar cómo están todos los datos separados cada uno en una línea y al final de cada línea nos dice a qué célula del mapa calibrado pertenece el dato y nos etiqueta el dato con la etiqueta, que tenía dicha célula si es que estaba etiquetada.

Ahora explicaremos un posible caso práctico donde pueda ser aplicado el algoritmo de clasificado y poder ver la importancia que tienen el algoritmo de entrenamiento y el algoritmo de calibrado que al final producen un mapa calibrado.

Imaginemos que la aplicación ha sido implantada en un hospital o ambulatorio. Si una persona acude al hospital y le hacen un análisis que recoja los niveles de los ocho valores que tiene cada dato del fichero diabetes, a ese dato con el resultado del análisis podríamos aplicar el algoritmo de clasificado con el mapa calibrado que ya hemos obtenido anteriormente.

Al aplicar el algoritmo de clasificado nos devolverá el dato del análisis clasificado y nos dirá a qué célula pertenece y que etiqueta le corresponda. Si es un 0 es muy posible que el paciente no sea diabético y si nos devuelve un 1 en la etiqueta el paciente es muy posible que sea diabético.

Este algoritmo se puede aplicar a otros muchos ejemplos de enfermedades que con este procedimiento con un análisis podemos saber con mucha probabilidad si el paciente tiene algún tipo de enfermedad.

Por supuesto el algoritmo se puede aplicar a otros campos. Es muy utilizado también en estudios estadísticos.

En la pantalla de clasificado si pulsamos el botón “Volver” la aplicación nos llevará a la pantalla de operaciones.

4. Diseño de la aplicación

El diseño de la aplicación se ha realizado en base a los requisitos funcionales recogidos en la fase de análisis, que están recogidos en el anexo de la memoria (pag. 57).

En este anexo se encuentran también los casos de uso de la aplicación (pag. 65), en base a los cuales se realiza la implementación de las distintas partes de las que consta la aplicación.

Por último, en el anexo encontramos el plan de pruebas (pag. 74), para una vez terminada la fase de implementación, se pueda comprobar el buen funcionamiento de todas las funcionalidades de la aplicación.

En este capítulo vamos a explicar el diseño de la aplicación mediante la estructura de paquetes de la forman. Para ello iremos explicando la función y el contenido de cada uno de los paquetes más importantes. También listaremos las clases java que componen el paquete explicando cual es la función de cada clase.

4.1 Paquete Actions

En este paquete se encuentran las clases que componen el controlador de la aplicación. El controlador comprende la funcionalidad involucrada desde que se produce un evento por parte del usuario hasta que se redirige a una pantalla de la aplicación.

Por lo tanto los actions son las clases que se encargan de recibir una orden del usuario, ejecutará la lógica de negocio que resuelva la funcionalidad deseada y redirija al jsp que debe generar la interfaz resultante.

Las clases que componen el paquete actions son:

- InicioAction.java: Esta clase se ejecuta con la url de inicio de la aplicación y se encarga únicamente de redirigir a la página de inicio.

- CargaDatosAction.java: Clase que tiene por función cargar en memoria las rutas de los ficheros de entrenamiento y mapa inicial. Si todo se ha desarrollado de manera correcta redirige a la pantalla de carga de parámetros.

- CargaParametrosAction.java: La función principal de esta clase es cargar en memoria los parámetros de entrada al algoritmo de entrenamiento. Una vez validados los datos de entrada redirige a la pantalla de operaciones.

- OperarAction.java: Este action se encarga de redirigir a la pantalla de operaciones. Se encarga también de comprobar si antes de realizar el calibrado se ha entrenado el mapa y de comprobar si antes de clasificar un fichero de datos se ha calibrado el mapa.

- EntrenarAction.java: En esta clase se realiza la acción de entrenar. Para ello lo primero que hace esta clase es leer el fichero de entrenamiento y volcar los datos en memoria. A continuación si el usuario seleccionó un fichero con el mapa inicial se lee también este fichero y se carga en la estructura de datos del mapa inicial. Si no lo seleccionó se inicializa un mapa inicial y se carga en memoria dicho mapa inicial.
  Si todo se ha realizado correctamente se procede a llamar al servicio de entrenamiento que es el encargado de ejecutar el algoritmo pasándole los mapas y los parámetros de entrada.
  Una vez terminado el algoritmo de entrenamiento el action redirige a la pantalla de resultado de entrenamiento. Si se ha producido algún error el action enviará al jsp con el mensaje del correspondiente error. En caso de haber terminado correctamente el action enviará al jsp los enlaces del mapa entrenado y del mapa inicial que ha guardado previamente en el servidor, además del grafico de la variable "J".

- CalibrarAction.java: Esta clase tiene como función principal realizar el algoritmo de calibrado para lo cual llamará al servicio de calibrado pasándole el mapa entrenado y los datos del fichero de datos elegido por el usuario. Una vez terminado con éxito esta clase nos redirigirá a la pantalla de resultado de calibrado con el enlace al fichero que guarda previamente en el servidor con el resultado del mapa calibrado.

- ClasificarAction.java: La función principal de este action es la de realizar el clasificado llamando al servicio de calibrado enviando a éste el fichero de datos que haya elegido el usuario, además del mapa calibrado. Una vez terminado el clasificado de los datos el action nos redirige a la pantalla de resultado de clasificado enviando el enlace que ha guardado en el servidor del fichero con los datos clasificados.

- EditarFicherosAjaxAction.java: Esta clase Ajax se ejecuta cuando pinchamos en alguno de los enlaces que se muestran en las páginas de resultado de entrenamiento, calibrado o clasificado y se encarga de abrir un fichero de texto con los datos de los ficheros que hemos guardado en el servidor y mostrárselo al usuario.

4.2 Paquete Services

En el paquete services se encuentran los servicios que se encargan de ejecutar la lógica de negocio de la aplicación.

En nuestra aplicación por tanto tendremos tres servicios que se encargarán de ejecutar el algoritmo de entrenamiento, el algoritmo de calibrado y el algoritmo de clasificado.

Las clases que componen el paquete services son:

- EntrenarServiceImpl.java: Esta es la clase que se encarga de ejecutar el algoritmo de entrenamiento que explicaremos a fondo en el próximo capítulo. Tendrá como entrada el fichero de entrenamiento, el mapa inicial y los parámetros introducidos por el usuario y si se ejecuta de manera correcta devolverá el mapa entrenado así como una lista de distancias mínimas que el action de entrenar usará para componer el gráfico de la variable "J".

- CalibrarServiceImpl.java: La función de este servicio es ejecutar el algoritmo de calibrado, que será explicado en el próximo capítulo. El servicio tendrá como entrada el fichero de datos y el mapa entrenado y devolverá el mapa calibrado.

- ClasificarServiceImpl.java: Esta servicio ejecuta el algoritmo de clasificado que también será explicado a fondo en el próximo capítulo. Este servicio tendrá como entrada el fichero de datos a clasificar y el mapa calibrado y devolverá los datos clasificados.

4.3 Paquete Beans

En este paquete se encuentran los beans que son clases java que se utilizan para contener los objetos que vamos a utilizar para interactuar en la aplicación.

Las clases que componen el paquete beans son:

- DatosEntrada.java: Esta clase java tiene como función manejar los datos de entrada de la aplicación. Esta clase está compuesta de los datos que van a ser recogidos en las pantallas de carga de datos y carga de parámetros y que son

introducidos por el usuario de la aplicación. La clase está compuesta por los siguientes atributos:

  - dimXmapa: En este atributo se guardará la dimensión x del mapa inicial recogida en la pantalla de carga de parámetros.

  - dimYmapa: En este atributo se guardará la dimensión y del mapa inicial recogida en la pantalla de carga de parámetros.

  - numCiclos: Atributo que guarda el número de ciclos que se va a iterar el algoritmo de entrenamiento. Este campo es recogido en la pantalla de carga de parámetros,

  - tasaAprendizaje: Este atributo es utilizado para que guarde la tasa de aprendizaje inicial recogida en la pantalla de carga de parámetros.

  - tipoTasaAprendizaje: En este atributo guardaremos el tipo de variación de la tasa de aprendizaje recogido en la pantalla de carga de parámetros.

  - limiteVecindario: En este atributo se guardará el número entero que indica el límite de vecindario inicial recogido en la pantalla de carga de parámetros.

  - rutaFicheroEntrenamiento: En este campo guardaremos la ruta del fichero de entrenamiento seleccionado por el usuario en la pantalla de carga de datos.

  - rutaMapaInicial: En este campo guardaremos la ruta del fichero de del mapa inicial seleccionado por el usuario en la pantalla de carga de datos.

- Dato.java: Esta clase java tiene como función manejar los datos de los ficheros de entrenamiento o del fichero que utilizamos para el clasificado. Por cada fila del fichero tendremos un dato que lo mapearemos en una instancia de esta clase. La clase está compuesta por los siguientes atributos:

  - valores: En este campo guardaremos la colección de valores que representan las coordenadas del punto en el mapa. Dependiendo de la dimensión de los datos tendremos tantos valores.

    - id: Este campo guarda el identificador del dato.

    - celulaGanadora: identificador de la célula ganadora una vez ejecutado el ciclo del algoritmo de entrenamiento.

    - distancia: En este campo guardaremos la distancia desde el punto a la célula ganadora.

    - etiqueta: En este campo guardaremos la etiqueta del dato si es que la tiene.

- Celula.java: Esta clase java tiene como función manejar las células de los mapas de entrenamiento, calibrado y del mapa inicial. Cada fila del fichero se corresponde a una célula la cual mapearemos en una instancia de esta clase. La clase está compuesta por los siguientes atributos:

    - id: Identificador de la célula.

    - dato: La clase célula extiende de Dato, por lo tanto la célula heredará también los campos de la clase dato.

- Mapa.java: Esta clase tiene como función guardar la colección de células que forman un mapa. La clase está compuesta únicamente por el siguiente atributo:

    - células: En este campo guardamos la colección de células que forman el mapa.

- DatosFichero.java: Esta clase tiene como función guardar los ficheros de datos de entrenamiento y el fichero que se utiliza para clasificar. Cuando leemos alguno de estos fichero mapeamos los datos en la estructura de esta clase. La clase está compuesta por los siguientes atributos:

    - dimensión: En este campo guardamos la dimensión de los datos del fichero que se corresponde con el dato que viene en la primera fila del fichero.

    - datos: Es este campo que es una colección de tipo Dato vamos guardando los datos del fichero.

- RespuestaEntrenar.java: En esta clase mapeamos el resultado del algoritmo de entrenamiento. Una vez ejecutado el algoritmo guardamos en este campo el mapa entrenado y la lista de distancias que devuelve el servicio al action. La clase está compuesta por los siguientes atributos:

- mapaEntrenado: En este campo guardamos la colección de células que forman el mapa entrenado.

- listaDistanciasMin: En este campo guardaremos la lista de distancias mínimas que devuelve el algoritmo de entrenamiento en cada ciclo. Esta lista de distancias serán las que formen luego el grafico de la variable "J".

4.4 Paquete Utils

Este paquete está formado por las clases de utilidades que pueden ser utilizadas en distintas partes de la aplicación. Por ello se agrupan en este paquete y son accesibles desde cualquier parte de la aplicación.

Las clases que componen el paquete utils son:

- Fichero.java: En esta clase agrupamos todos los métodos que tienen que ver con la interacción con ficheros. Alguno de los métodos más importantes son:

    - leerFichero: Con este método leemos los datos de un fichero de datos. Le pasamos la ruta del fichero que queremos leer y nos devuelve los datos en un objeto de la clase DatosFichero.java explicada anteriormente.

    - leerMapaInicial: Mediante este método leemos los datos del fichero del mapa inicial. Se le pasa la ruta del fichero que contiene el mapa y devuelve los datos del mapa en un objeto de la clase Mapa.java explicada anteriormente.

    - escribirFicheroClasificado: Este método recibe los datos que ya han sido clasificados y los graba en un fichero de texto que los guarda en un fichero temporal del servidor.

    - escribirFicheroMapa: Este método graba en un fichero de texto el mapa que se le pasa como parámetro. Este método lo utilizamos para guardar en un fichero temporal del servidor los mapas de entrenamiento e inicial.

    - escribirFicheroDistanciasMinJ: Este método escribe en un fichero de texto que guarda en el servidor la lista de las distancias mínimas que nos devuelve el algoritmo de entrenamiento.

- Constantes.java: En esta clase se agrupan todas las constantes que son utilizadas en la aplicación. La finalidad de esta clase es que si en algún momento el valor de la constante debe cambiar solo habría que hacerlo una vez y no en todos los sitios donde se utiliza la constante.

- Utiles.java: En esta clase agrupamos distintos métodos que son utilizados en la aplicación. Estos métodos son de todo tipo y pueden realizar cualquier tipo de función. Algunos de estos métodos son:

    - calculaDistanciaEuclidea: Método que calcula la distancia euclídea que la usamos para calcular la distancia entre dos puntos,

    - calculaAprendizaje: Este método realiza el aprendizaje a una célula respecto a un punto aplicando una tasa de aprendizaje.

    - etiquetaMayoritariaCelula: Este método nos dice cuál es la etiqueta que más se repite en una célula.

    - obtenerCelulasVecinas: Este método nos devuelve las células vecinas en el mapa.

5. Desarrollo del sistema

En este capítulo se explican cómo se han desarrollado las distintas partes de la aplicación, haciendo especial atención a los algoritmos más importantes.

5.1 Enfoque de la aplicación

Uno de los objetivos de la aplicación es realizar una aplicación que sirva de aprendizaje a los alumnos. Por esta razón se decidió realizar una aplicación web con el lenguaje de programación java.

¿Por qué una aplicación web? Tomamos esta decisión por distintas razones.

Al ser una aplicación didáctica que iba a ser utilizada por los alumnos de una clase una de las razones que nos llevaron a tomar la decisión de diseñar una aplicación web es que ésta, la aplicación web es instalada en un único servidor y de esta manera solo tiene que ser instalada una vez independientemente de los usuarios que la vayan a utilizar, además de que las aplicaciones web permiten múltiples usuarios de manera concurrente. No hay que instalarla en cada terminal del usuario como sucede con otras aplicaciones que hay que instalarla tantas veces como terminales hayan. Por lo tanto la aplicación es mucho más mantenible y los costes de instalación y mantenimiento son escasos en comparación a otros tipos de aplicaciones ya que todo se centraliza en un servidor.

Además, cada vez que haya que realizar una actualización de la aplicación solo hay que realizarla una vez, subiendo al servidor el nuevo ejecutable de la aplicación.

Únicamente el terminal del usuario tiene que tener conexión a internet y un buen navegador web, y ambos requisitos se tienen en las aulas de informática de la universidad. Cuando el alumno inicie la sesión en el pc del aula solo tendrá que abrir el navegador y escribir la url de inicio de la aplicación y esta se ejecutará en el pc del alumno el cual podrá comenzar a realizar la práctica, de esta manera tan rápida sin necesidad de tener que descargar la aplicación ni de instalarla ni configurarla como ocurre con otros tipos de aplicaciones.

En definitiva, después de analizar los requisitos de la aplicación vimos que las ventajas de las aplicaciones web encajaban con las características que tenía que tener nuestra aplicación.

5.2 Carga de datos y parámetros.

En las pantallas de carga de datos y carga de parámetros la aplicación recoge los parámetros que van a servir de entrada para la ejecución del algoritmo de entrenamiento.

Ambas pantallas contienen dos formularios donde el usuario va a introducir los parámetros que considere necesarios para ejecutar el algoritmo.

En la pantalla de carga de datos el formulario que se muestra al usuario contiene dos seleccionables de ficheros donde podrá seleccionar las rutas del fichero de entrenamiento y del fichero inicial. Y en la pantalla de carga de parámetros se muestra un formulario donde el usuario podrá introducir los parámetros de entrada al algoritmo de entrenamiento que son: las dimensiones "x" e "y" del mapa, el número de ciclos que se ejecutará el algoritmo, la tasa de aprendizaje inicial y tipo de variación de la tasa de aprendizaje.

En ambas pantallas, una vez el usuario pulse el botón "Cargar Datos" la aplicación ejecutará el action correspondiente y guardará los parámetros de entrada en memoria en un objeto de la clase java DatosEntrada.

La clase DatosEntrada contiene los atributos necesarios para guardar los datos y parámetros de entrada y se guardarán en memoria hasta que se envíen los datos como entrada para la ejecución del algoritmo de entrenamiento.

5.3 Algoritmo de entrenamiento

Una vez tenemos los parámetros de entrada que el usuario ha elegido comenzamos el entrenamiento.

Para ello lo primero que tenemos que hacer es cargar los datos de entrenamiento. En los datos de entrada el usuario seleccionó la ruta del fichero de entrenamiento. La aplicación carga en un objeto de la clase DatosFichero.java los datos del fichero de entrenamiento, donde guardamos la dimensión del fichero y la colección de datos. Para ello usamos el método leerFichero de la clase Fichero.java.

Este método lee la primera línea para obtener la dimensión de los datos del fichero y de esta manera crear los objetos de datos con esta dimensión. Luego con un bucle hasta final de fichero leemos el resto de líneas con los datos, hasta que tengamos cargados los datos en una colección. Si en algún momento el método detecta algún error de formato del fichero devuelve error y se para el algoritmo de entrenamiento devolviendo el error correspondiente al usuario por pantalla.

Para la lectura del fichero hemos utilizado las clases FileReader y BufferReader de la librería java.io.

Una vez cargados los datos de entrenamiento el siguiente paso es inicializar el mapa inicial. Para inicializar el mapa tenemos dos opciones.

Si el usuario seleccionó la ruta del fichero con el mapa inicial la aplicación cargará el mapa seleccionado en un objeto de la clase Mapa.java donde se guardara la colección de células (Celulas.java). El mapa se carga en memoria mediante el método leerMapaInicial de la clase Fichero.java. Este método se encarga de leer el mapa inicial seleccionado por el usuario y lo carga en memoria de forma parecida a como lo hicimos para leer los datos de entrenamiento.

Si el usuario no seleccionó ningún fichero se genera un mapa inicial. El número de células que tendrá el mapa viene determinado por la dimensión x y la dimensión y que relleno el usuario por pantalla en la carga de parámetros. Multiplicando ambas dimensiones tendremos el número de células que tendrá nuestro mapa inicial. Mediante un bucle que se ejecutará tantas veces como células tengamos que crear, crearemos las nuevas células con la dimensión que tienen los datos de entrenamiento. Una vez terminado el bucle tenemos el mapa inicial cargado en memoria.

En este momento se invoca al algoritmo de entrenamiento donde le pasamos los datos del fichero de entrenamiento, el mapa inicial y los parámetros recogidos por pantalla.

El algoritmo de entrenamiento es un bucle que se ejecutará tantas veces como indique el número de ciclos que introdujo el usuario por pantalla.

En cada ciclo por cada dato obtendremos su célula más cercana y realizaremos el aprendizaje de la célula ganadora en dirección al punto. Para ello tendremos un nuevo bucle que ejecutaremos tantas veces como datos de entrenamiento tengamos.

Para cada dato tenemos que obtener cuál es la célula más cercana. Para ello tendremos un nuevo bucle donde iremos recorriendo las células de mapa inicial y calcularemos la distancia desde el punto a cada célula. Para calcular la distancia usamos la distancia euclídea que obtenemos mediante el método calculaDistanciaEuclidea de la clase Utiles.java qué se encarga de obtener la distancia euclídea desde el punto a la célula.

Una vez terminado el bucle de las células tendremos la célula más cercana al dato, es decir la célula ganadora. Apuntaremos en el dato cuál es su célula más cercana y la distancia a esta.

El paso siguiente es realizar el aprendizaje de la célula ganadora al dato que lo haremos mediante el método calculaAprendizaje de la clase Utiles.java. A este método le pasamos el dato, la célula y la tasa de aprendizaje. La tasa de aprendizaje

fue introducida por el usuario en la carga de parámetros pero no se aplica con el mismo valor en cada ciclo ya que en los primeros ciclos será mayor puesto que se irá decrementando su valor en cada ciclo, siendo muy pequeña en los últimos ciclos.

La tasa de aprendizaje varía en función del tipo de variación de la tasa de aprendizaje que haya elegido el usuario en la carga de parámetros. El porcentaje que se irá decrementando en cada ciclo, y lo calculamos mediante el método getTasaAprendizaje de la clase Utiles.java que en función del tipo de tasa elegido (sin variación, lineal, logarítmica) aplica la fórmula correspondiente. Si es sin variación aplicamos siempre la misma tasa de aprendizaje, si no, la tasa será decrementada según lo que nos devuelva el método getTasaAprendizaje.

El tipo de tasa de aprendizaje lineal calcula el porcentaje dividiendo la tasa de aprendizaje entre el número de ciclos. Mientras que si el tipo de tasa elegido es el logarítmico la función que aplica es: tasa_inicial * exp (-p * c / total_ciclos).

El método calculaAprendizaje aplica la función de aprendizaje con la tasa de aprendizaje calculada y acercar la célula hacia el punto, devolviendo la nueva posición de la célula. A continuación modificamos el mapa cargado en memoria con la nueva situación de la célula.

El siguiente paso es calcular el aprendizaje de las células vecinas. Para ello lo primero que tenemos que obtener son las células vecinas. Para ello cogemos del mapa todas las células vecinas de la célula ganadora con un límite de vecindario que fue indicado por el usuario en la carga de parámetros. El método obtenerCelulasVecinas de la clase Utiles.java se encarga de obtenerlas células vecinas. Al parámetro le pasamos la célula ganadora y el mapa que estamos entrenando. El método se encarga de recorrer el mapa para obtener las células vecinas que se encuentran en el mapa a una distancia marcado por el límite de vecindario. El método nos devuelve una colección con las células vecinas.

El cálculo del aprendizaje de las células vecinas lo hacemos de igual forma que hicimos con la célula ganadora pero aplicando la tasa de aprendizaje de manera que se decremente según el nivel de vecindad de las células. Las células vecinas de primer nivel son las que están a su lado en el mapa. El nivel va avanzando según vamos recorriendo el mapa desde la célula, las de segundo nivel estarán a dos pasos desde la célula ganadora. Por lo que la fórmula que utilizamos para la tasa de aprendizaje es dividirla por 1 + el nivel de la célula, siendo la tasa de aprendizaje menor para los niveles mayores, por lo que las células se irán acercando menos al punto cuanto más se alejen de la célula ganadora.

Una vez obtenemos la célula ganadora de todos los datos de entrenamiento salimos del bucle ya que hemos terminado el primer ciclo. Cuando terminamos un ciclo hacemos dos cosas: calculamos la suma de las distancias mínimas que son las distancias de cada punto a su célula ganadora y añadimos la distancia total a una colección que tendrá la suma de las distancias mínimas en cada ciclo. La otra cosa

que hacemos es actualizar la tasa de aprendizaje, restándola el porcentaje calculado anteriormente para ser utilizada en el siguiente ciclo.

El algoritmo se sigue ejecutando hasta que termine el número de ciclos que se dará por terminado el algoritmo de entrenamiento.

Una vez terminado el algoritmo ya tenemos el mapa entrenado con las nuevas posiciones de las células después del entrenamiento y lo guarda en memoria para ser utilizado en otras operaciones.

La aplicación escribe en un fichero de texto la lista con las distancias mínimas y lo guarda en el servidor y guardando la ruta al fichero para mostrarla al usuario y que este pueda descargarlo si lo desea, al igual que hace con el mapa inicial y el mapa entrenado.

Antes de que la aplicación muestre la página con los resultados del entrenamiento, si todo va bien, prepara dos arrays JSON. En uno de ellos cargamos la lista de distancias mínimas y en otra los ciclos que se ha ejecutado. Estos dos arrays se envían a la página de resultado para poder crear el grafico con la evolución de la J, la lista de distancias mínimas. Estos arrays son necesarios para crear el gráfico con la librería utilizada, jfreechart.

Si durante el transcurso del algoritmo se produjera algún error, este se pararía y mostraría en la pantalla de resultado entrenamiento el correspondiente error.

5.4 Algoritmo de calibrado

El algoritmo de calibrado consiste en etiquetar las células del mapa entrenado. Las células se etiquetan en función de un fichero de datos que el usuario selecciona por pantalla. En este fichero cada dato puede o no tener una etiqueta además de los valores del dato y son estas etiquetas las que nos dirán cómo una célula queda etiquetada.

En primer lugar tenemos que leer el fichero de datos elegido por el usuario o si este no lo seleccionó utilizaremos el fichero de datos de entrenamiento. Para leer el fichero utilizamos el método de lectura de ficheros explicado anteriormente.

Una vez tengamos cargados los datos del fichero en una instancia de la clase DatosFichero la enviaremos como parámetro de entrada del algoritmo junto con el mapa entrenado que tenemos guardado en memoria.

Si no se ha realizado el entrenamiento antes no tendremos cargado en memoria el mapa entrenado y por consiguiente no se podrá realizar el calibrado hasta que se realice en entrenamiento. Esto ocurre cuando el usuario pincha en la pantalla de

operaciones la opción “Calibrar” sin haber realizado el entrenamiento antes y por consiguiente mostramos por pantalla el mensaje informando a usuario.

Para realizar el calibrado se ejecuta un algoritmo mediante el cual se recorren los datos del fichero y por cada dato se busca cual es la célula más cercana. Si un dato no tiene etiqueta directamente pasamos al siguiente dato puesto que no nos interesa para calibrar el mapa. Una vez sepamos la célula ganadora miraremos la etiqueta del dato y apuntaremos que esa célula tiene una coincidencia con esa etiqueta.

Para ello hemos creado una estructura de datos donde tenemos un mapa (HashMap clase de la libraría java.util) donde cada entrada del mapa es un par, con el identificador de cada célula y asociado a este una colección de etiquetas. Entonces cuando tenemos una célula ganadora y una etiqueta de un dato el procedimiento será el siguiente: iremos al mapa con el id de la célula ganadora, si no está la célula en el mapa la añadiremos y en su colección de etiquetas añadiremos el nombre de la etiqueta del dato. Si la célula ya había salido ganadora antes y está añadida al mapa obtendremos la colección de etiquetas de dicha célula y añadiremos esta etiqueta.

Así con todos los datos del fichero y una vez terminados procedemos al conteo de las etiquetas de cada célula.

El mapa entrenado y el mapa calibrado tienen la única diferencia de que en el mapa calibrado las células tienen una etiqueta, por lo que haremos una copia del mapa entrenado como principio de nuestro mapa calibrado. Por lo tanto iremos recorriendo las células del mapa clonado y por cada célula nos iremos al mapa que hemos ido rellenado durante el algoritmo, y obtendremos la colección de etiquetas que tiene dicha célula. Haremos un conteo de las etiquetas de cada célula y veremos cuál es la que más se repite y entonces iremos al mapa que hemos clonado y en la célula correspondiente pondremos la etiqueta que más se ha repetido en el conteo.

Lo haremos así por cada célula y al final del algoritmo tendremos el mapa clonado del algoritmo de entrenamiento con las células etiquetadas, es decir, tendremos nuestro mapa calibrado.

Una vez terminado correctamente el algoritmo de calibrado la aplicación guarda el resultado del fichero con el mapa calibrado al igual que hicimos con el fichero del mapa entrenado, mostrando en la pantalla de resultado de calibrado el enlace al fichero guardado en el servidor.

5.5 Algoritmo de clasificado

En primer lugar para ejecutar el algoritmo de clasificado necesitamos el fichero de datos que queremos clasificar. El usuario en la pantalla de clasificado seleccionará el fichero y se cargará en memoria en una instancia de la clase DatosFichero explicada anteriormente. Si el usuario no seleccionara ningún fichero el algoritmo se ejecutaría con el fichero de datos de entrenamiento.

Una vez tengamos el fichero de datos cargado en memoria se ejecuta el algoritmo pasándole como parámetros de entrada la estructura de datos con los datos del fichero a clasificar y el mapa calibrado.

El algoritmo de clasificado es bastante sencillo. Consiste en ir iterando los distintos datos del fichero que queremos clasificar y por cada dato vamos a obtener la célula más cercana al dato del mapa calibrado, que va a ser la célula a la que pertenece el dato..

Una vez tengamos la célula ganadora, la célula más cercana al dato, le asignaremos al dato la célula a la que pertenece y la etiqueta que tiene dicha célula.

Por lo tanto tendremos una colección de datos con una etiqueta y la célula a la que pertenece. Ya solo nos faltará pasar esta colección de datos a un fichero y tendremos el fichero de datos clasificado donde podremos ver para cada dato, la célula a la que pertenece y la etiqueta que tiene.

Una vez acabado el algoritmo correctamente la aplicación guardará el fichero de datos en el servidor de aplicaciones y mostrará el enlace al fichero clasificado para que el usuario al pincharlo puede observar el fichero clasificado en formato .txt y pueda guardarlo si lo necesita,

5.6 Ficheros guardados en directorio temporal en servidor

La aplicación nos requería guardar y mostrar los ficheros que contiene el mapa inicial, el mapa de entrenamiento, el mapa calibrado o el fichero de datos clasificado donde el usuario podría comprobar el resultado de los algoritmos.

Al ser una aplicación web que se ejecuta en un servidor externo, no tiene relación con la máquina desde donde el usuario está ejecutando la aplicación y por lo tanto el servidor web no tiene manera de acceder a la máquina del usuario o por lo menos una forma eficaz de hacerlo. A diferencia de una aplicación que se ejecutara directamente en el pc del usuario donde podría tener acceso a los directorios y almacenar los ficheros en alguna ruta determinada dentro de la propia máquina.

Por lo tanto la solución que tomamos para resolver este problema fue crear un algoritmo que genera un fichero de datos y lo guarda con formato .txt en un directorio temporal del servidor web. La ruta donde se guardan los ficheros será en el directorio "/temp" que colgará del directorio donde este el ejecutable de la aplicación. El encargado del mantenimiento del servidor tendrá que tener en cuenta que en este directorio se irán acumulando ficheros de los distintos alumnos que realicen la práctica por lo que se tendrá que realizar por procedimiento un borrado de ficheros cada cierto tiempo, el que se considere oportuno, con el fin de evitar un consumo elevado de los recursos del servidor.

Las clases que hemos utilizado para realizar la escritura de ficheros son las clases FileWriter, BufferedWriter que son propias del api de java y se encuentran en la librería java.io importada previamente en la aplicación.

Una vez guardados los ficheros de datos, la aplicación mostrará por pantalla al usuario un enlace al directorio temporal del servidor web donde se encuentra el fichero. Cuando el usuario pincha en alguno de los enlaces con el resultado de los algoritmos, la aplicación abre al usuario por pantalla un fichero de datos mostrándole el resultado de los mismos.

Para realizar esta acción se ejecuta una clase propia de nuestra aplicación, la clase EditarFicherosAjaxAction que al ser una clase Ajax sin desviarnos del flujo de nuestra aplicación abrirá el fichero de datos al usuario. Esta clase se apoya de dos clases fundamentales para generar el fichero y mostrarlo por pantalla que son la clase File y la clase Desktop que se encuentran en las librerías java.io y java.awt que tenemos importadas en nuestra aplicación. Mediante la clase File generamos el fichero a partir de una ruta, la ruta del servidor web donde se encuentra almacenado el fichero, y mediante la clase Deskpot abrimos el archivo de la clase File creado anteriormente con la función "open" propia de la clase Deskpot que será la encargada de mostrar por pantalla el fichero de datos .txt al usuario.

De esta manera se ha solucionado el problema de mostrar los ficheros al usuario y una vez éste visualice el fichero también podrá tener la opción de guardarlo en su propia máquina a través de la interfaz que ofrece el gestor de fichero que tenga el usuario predeterminado en su máquina para abrir ficheros .txt.

6. Conclusiones y líneas futuras

Una vez terminada la aplicación podemos decir que hemos cumplido el objetivo del que partíamos cuando empezamos con el proyecto.

Se ha realizado una aplicación que podrá ayudar a los alumnos de informática de la universidad a comprender mejor los mapas autoorganizados y las redes de neuronas y que podrá ser instalada de una manera sencilla en un servidor web de la universidad para que los alumnos desde las aulas de informática puedan realizar la práctica, simplemente ejecutando la url de la aplicación en un navegador web, sin necesidad de descargarse ni instalar ningún programa.

Mediante la aplicación podrán comprobar con casos prácticos cómo funcionan los mapas autoorganizados de Kohonen, pudiendo comprobar in situ como se realiza el entrenamiento de estos mapas auto-organizados a partir de datos reales. Al realizar el entrenamiento del mapa, se puede observar cómo funciona el algoritmo, mejorando en cada ciclo, pudiéndose observar en el gráfico de las distancias mínimas, cómo la distancia va disminuyendo en cada ciclo.

El usuario en todo momento podrá interactuar con la aplicación pudiendo utilizar ficheros de datos elegidos por el usuario, introduciendo distintos parámetros de entrada para el algoritmo de entrenamiento y también pudiendo tratar con los distintos ficheros de resultados que la aplicación ofrece al usuario mediante enlaces en las pantallas de resultados.

El alumno podrá comprender también para qué sirve el calibrado de un mapa auto-organizado, generando el mapa calibrado, aprovechando de esta forma la información que se pueda obtener cuando se dispone de datos etiquetados.

También podrá comprobar cómo funciona la clasificación, utilizando datos de entrada de casos reales que reciben una etiqueta, ejecutando el algoritmo de clasificación con los datos y el mapa calibrado.

Se ha realizado una aplicación que cumple con todos los requisitos que creíamos necesarios para realizar este proyecto pero se me han quedado algunas ideas pendientes para próximas evoluciones de esta aplicación que con gran ilusión las haría en el proyecto fin de carrera del grado.

Alguna de estas ideas es perfeccionar el algoritmo de aprendizaje añadiendo un parámetro para elegir una función de vecindario. Y además de añadir alguna idea que tenga el departamento, sobre todo elegir un buen programa gráfico que sea compatible con nuestra aplicación web y que pueda mostrar gráficamente el mapa en el espacio n-dimensional y la evolución de dicho mapa durante el entrenamiento.

7. Bibliografía

- G. Alwang, J. Clyman, R.V. Dragan, L. Seltzer: Java Guide. PC Magazine, vol.17 no. 7 (April 1998), p. 101-209.

- G. Alwang, J. Clyman, R.V. Dragan, L. Seltzer: Java Guide. PC Magazine, vol.17 no. 7 (April 1998), p. 101-209.

- P.M. Cuenca Jiménez: Programación en JAVA para Internet. Anaya Multimedia (col. Vía@Internet), Madrid, 1996. xxiv+528p.

- R.V. Dragan, L. Seltzer: Java: A Field Guide for Users. PC Magazine, vol.16 no. 10 (May 1997), p. 100-115.

- J.M. Framiñan Torres: Java. Anaya Multimedia (col. Al Día en una Hora), Madrid, 1997. 128p.

- R. Gilbertson: NC's the Fourth Wave — The Network computer is switching the client-server standard to server-client. Tech Web News (June 16, 1997).

- J. Manger: Java: More than just a programming language. Internet Business Magazine (April 1997), p. 66-70.

- Luján Mora, Sergio (2001). Programación en Internet: Clientes Web (libro completo gratuito en pdf) (1ª edición). Editorial Club Universitario.

- Programación de aplicaciones web: historia, principios básicos y clientes web (libro completo gratuito en pdf) (1ª edición). Editorial Club Universitario.

- P. Jones: Java and Libraries. Digital and Otherwise. D-Lib Magazine (March 1997). [Se trata de un artículo de una revista electrónica, que puede consultarse p. ej. en http://mirrored.ukoln.ac.uk; su versión impresa ocupa unas 5p.]

- http://www.javasoft.com, http://www.sun.com —Sitios oficiales de Sun Microsystems.

- http://www.javaworld.com/JavaWorld —Revista electrónica sobre Java.

- "Gamelan: The Java Directory" en http://java.developer.com (también puede invocarse con http://www.gamelan.com) —Gamelan es uno de los más conocidos recursos en Internet sobre Java; contiene listas anotadas, bien organizadas y actualizadas de miles de applets.

- http://www.iii.com —Innovative Interfaces y http://www.sls.se —SLS (Information Systems), División Europea de Innovative Interfaces.

- Isasi, P., Galván, I.M., Redes de Neuronas Artificiales. Un enfoque práctico. Pearson 2004.

- Valls, J. M., Galván, I. M. (15/09/2011), Material de Clase. Tema 5 [consulta: 10-09-2014], desde el sitio Web de OCW - UC3M: Disponible en: http://ocw.uc3m.es/ingenieria-informatica/redes-de-neuronas-artificiales/transparencias/material-de-clase.-tema-5.

- Kohonen, Teuvo et al. Chapter 8. Self-organizing map. Biennial Report 2002-2003. Laboratory of Computer and Information Science. Neural Networks Research Centre. Helsinki University of Technology. P.O. Box 5400. FI-02015 HUT, Finland. K.

- Teuvo Kohonen, Dr. Eng., Emeritus Professor of the Academy of Finland; Academician. Neural Networks Research Centre, Laboratory of Computer and Information Science. [consulta: 10-09-2014]. Disponible en http://users.ics.aalto.fi/teuvo/index.shtml.

- Kohonen, Teuvo et al. SOM_PAK: The Self-Organizing-Map Program Package. SOM Programming Team of the Helsinki University of Technology. Otaniemi, enero de 1996. [consulta: 10-09-2014]. Disponible en http://hackbbs.org/article/reds/form_de_base/SBI/labo5/som_pak.pdf

- SOM_PAK AND LVQ_PAK. Neural Networks Research Centre. Laboratory of Computer and Information Science. Helsinki University of Technology. [consulta: 13-09-2014]. Disponible en: http://www.cis.hut.fi/research/som-research/nnrc-programs.shtml.

- http://www.cis.hut.fi/research/som_pak/som_doc.txt

- Welcome To JFreeChart!. [consulta: 10-09-2014]. Disponible en: http://www.jfree.org/jfreechart/.

- jfree.org. (s.f.). JFreeChart - API. Recuperado el 2013, de http://www.jfree.org/jfreechart/api/javadoc/

- jfree.org. (s.f.). JFreeChart - API clase ChartPanel. Recuperado el 2013, de http://www.jfree.org/jfreechart/api/gjdoc/org/jfree/chart/ChartPanel.html

- blogspot, m. . (s.f.). Tutorial JFreeChart. Recuperado el 2013, de http://monillo007.blogspot.com/2011/12/hacer-graficas-con-java.html

- España. Jefatura del Estado. Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal. Última modificación: 5 de marzo de 2011. [consulta: 12-09-2014]. Disponible en: http://www.boe.es/buscar/pdf/1999/BOE-A-1999-23750-consolidado.pdf.

- blog.iedge.eu. (s.f.). ciclo de vida desarrollo software. Recuperado el 2013, de http://blog.iedge.eu/wp-content/uploads/2011/09/IEDGE-ciclo-de-vida-desarrollo-software-2.jpg

8. Diccionario de términos

- Ajax: AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

- Algoritmo: es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad

- Aplicación: es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos.

- Arquitectura: es el diseño conceptual y la estructura operacional fundamental de una aplicación. Es decir, es un modelo y una descripción funcional de los requerimientos y las implementaciones de diseño del programa construido.

- Array: En programación, es una zona de almacenamiento continuo, que contiene una serie de elementos del mismo tipo, los elementos de la matriz. Desde el punto de vista lógico una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones). (Wikipedia, Vector (informática))

- Bucle: es una sentencia que se realiza repetidas veces a un trozo aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse.

- Cluster: se aplica a los conjuntos o conglomerados de neuronas unidos entre sí normalmente por una red de alta velocidad y que se comportan como si fuesen una única computadora

- Conexión: Punto donde se realiza el enlace entre aparatos o sistemas. En este caso particular de dos neuronas situadas en dos capas n y n+1.

- Entrada: se refiere a la información recibida en un mensaje, o bien al proceso de recibirla.

- Formulario: permite al usuario introducir datos los cuales son enviados a un servidor para ser procesados. Los formularios web se parecen a los formularios

de papel porque los internautas llenan dichos formularios usando casillas de selección, botones de opción, o campos de texto.

- Funcionalidad: Conjunto de características que hacen que un método o conjunto de métodos, de una o varias clases, sea práctico. (WordReference)

- Interfaz: Conexión física y funcional entre dos sistemas o dispositivos de cualquier tipo dando una comunicación entre distintos niveles. (Wikipedia, Interfaz).

- Internet: es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, lo cual garantiza que las redes físicas heterogéneas que la componen funcionen como una red lógica única de alcance mundial.

- Intranet: es una red informática que utiliza la tecnología del Protocolo de Internet para compartir información, sistemas operativos o servicios de computación dentro de una organización.

- Jfreechart: es un marco de software open source para el lenguaje de programación Java, el cual permite la creación de gráficos1 complejos de forma simple.

- Json: es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

- Lenguaje orientado a objetos: Forma de denominar a aquellos lenguajes de programación que implementan los conceptos definidos por la programación a objetos.

- Mapa: está formado por una red de neuronas que se encuentra en un espacio n-dimensional.

- Multiplataforma: es un atributo conferido a programas informáticos o métodos y conceptos de cómputo que son implementados e interoperan en múltiples plataformas informáticas.

- Navegador: es un software, aplicación o programa que permite el acceso a la Web, interpretando la información de distintos tipos de archivos y sitios web para que estos puedan ser visualizados.

- Neurona: Es un punto de intersección o unión de dos o más elementos. En este caso particular están además situados en capas.

- Parámetro: es una variable que es enviada al invocar una función y que forma parte de los lenguajes de programación.

- Proceso: puede informalmente entenderse como un programa en ejecución. Formalmente un proceso es "Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados".

- Ruta: Es la forma de referenciar un archivo informático o directorio en un sistema de archivos de un sistema operativo determinado. Una ruta señala la localización exacta de un archivo o directorio mediante una cadena de caracteres concreta. (Wikipedia, Ruta / Path).

- Salida: es el proceso de transmitir la información por un objeto (el uso de verbo). Esencialmente, es cualquier dato que sale de un sistema.

- Servidor web: es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente.

ANEXO 1. Gestión del proyecto

1 Requisitos funcionales

En este apartado se indicarán los requisitos de usuario que se han tenido en cuenta para desarrollar la aplicación.

REQUISITO FUNCIONAL (RF-01: Pantalla inicial)
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja** **Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja** **Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable
DESCRIPCION
Una vez que ejecuta la aplicación se mostrará una página de bienvenida con un menú que tendrá dos opciones: - Inicio práctica: Pinchando en esta opción del menú comenzará la práctica y se mostrará la primera pantalla, la carga de datos. - Salir: Esta opción del menú provoca la salida de la aplicación cerrando la aplicación abierta por el usuario.

Tabla 5. RF-05

REQUISITO FUNCIONAL (RF-02: Carga fichero de entrenamiento)
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja** **Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja** **Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable
DESCRIPCION
La aplicación debe permitir al usuario a través de la pantalla de carga de datos seleccionar un fichero de entrenamiento guardado en el PC del usuario. Este campo debe pertenecer a un formulario de la pantalla de carga de datos y debe ser obligatorio.

Tabla 6. RF-06

REQUISITO FUNCIONAL (RF-03: Carga fichero del mapa inicial)

Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:** ☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La aplicación debe permitir al usuario a través de la pantalla de carga de datos seleccionar un fichero que contenga el mapa inicial guardado en el PC del usuario. Este campo debe pertenecer a un formulario de carga de datos y debe ser opcional.	

Tabla 7. RF-07

REQUISITO FUNCIONAL (RF-04: Carga de datos y paso a carga de parámetros)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:** ☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La pantalla de carga de datos deberá contener un botón "Aceptar" que al pulsarlo valide que se ha seleccionado el fichero de entrenamiento. De no haberse seleccionado el fichero de entrenamiento la aplicación deberá mostrar un mensaje de validación al usuario advirtiéndole que para cargar los datos y pasar a la siguiente pantalla debe seleccionarlo. Si el fichero se ha seleccionado y se ha pulsado el botón "Aceptar" la aplicación nos llevará a la pantalla de carga de parámetros.	

Tabla 8. RF-08

REQUISITO FUNCIONAL (RF-05: Botón limpiar)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:** ☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La aplicación debe contener en todos los formularios un botón "Limpiar" que borre todos los campos modificados en cada formulario.	

Tabla 9. RF-09

REQUISITO FUNCIONAL (RF-06: Dimensión x del mapa)

Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La pantalla de carga de parámetros debe recoger la dimensión x del mapa inicial. Este campo será una caja de texto y deberá recoger un número entero que será obligatorio.	

Tabla 10. RF-010

REQUISITO FUNCIONAL (RF-07: Dimensión y del mapa)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La pantalla de carga de parámetros debe recoger la dimensión y del mapa inicial. Este campo será una caja de texto y deberá recoger un número entero que será obligatorio.	

Tabla 11. RF-011

REQUISITO FUNCIONAL (RF-08: Número de ciclos)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La pantalla de carga de parámetros debe recoger el número de ciclos que se va a ejecutar el algoritmo. Este campo será una caja de texto y recogerá un número entero y será obligatorio.	

Tabla 12. RF-012

REQUISITO FUNCIONAL (RF-09: Tasa de aprendizaje)

Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La pantalla de carga de parámetros debe recoger la tasa de aprendizaje. Este campo será una caja de texto y recogerá un número que admite decímales y será obligatorio.	

Tabla 13. RF-013

REQUISITO FUNCIONAL (RF-10: Variación de la tasa de aprendizaje)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuent0e:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La pantalla de carga de datos debe permitir al usuario seleccionar el tipo de variación de tasa de aprendizaje y para ello lo hará mediante un campo seleccionable que recoja los siguientes tipos de tasas de aprendizaje: - Variación lineal - Variación logarítmica - Sin variación	

Tabla 14. RF-14

REQUISITO FUNCIONAL (RF-11: Carga de parámetros y envío pantalla de operaciones)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
La pantalla de carga de parámetros deberá contener un botón “Aceptar” que al pulsarlo valide si todos los campos del formulario están rellenos, de no ser así se enviará un mensaje de validación al usuario advirtiéndole que debe rellenar todos los campos del formulario. En caso de estar todos los campos rellenos, al pulsar el botón “Aceptar” la aplicación cargará en memoria los parámetros de entrada para ser utilizados con posterioridad en los algoritmos y la aplicación redirigirá a la pantalla de operaciones.	

Tabla 15. RF-15

REQUISITO FUNCIONAL (RF-12: Operaciones)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
Una vez cargados los datos y los parámetros se pasará a la pantalla de operaciones. Esta pantalla tendrá cuatro botones a modo menú: - Volver: Pulsando este botón la aplicación nos llevará de nuevo a la pantalla de carga de parámetros. - Entrenar: Pulsando este botón la aplicación ejecutará el algoritmo de entrenamiento y nos enviará a la pantalla con el resultado de entrenamiento. - Calibrar: Mediante este botón la aplicación nos enviará a la pantalla de calibrado. Si no se ha realizado el entrenamiento antes de pulsar este botón la aplicación deberá mostrar un mensaje al usuario advirtiéndole que antes de realizar el calibrado tiene que realizarse el entrenamiento. - Clasificar: Al pulsar este botón la aplicación nos enviará a la pantalla de clasificado. Si no se ha realizado el calibrado de datos antes de pulsar este botón la aplicación deberá mostrar un mensaje al usuario advirtiéndole que antes de realizar el clasificado tiene que realizarse el calibrado.	

Tabla 16. RF-16

REQUISITO FUNCIONAL (RF-13: Entrenamiento)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
Una vez pulsado el botón “Entrenar” debe comenzar el entrenamiento. La aplicación aplicará el algoritmo aplicando los parámetros recogidos por pantalla en base al fichero de entrenamiento. Una vez terminado el entrenamiento la aplicación redirigirá a la pantalla de entrenamiento. Si se ha producido algún error durante el entrenamiento la aplicación lo deberá mostrar por pantalla. En caso contrario se mostrará que se ha ejecutado correctamente. La pantalla de entrenamiento debe contener un botón “Volver” que al pulsarlo nos envíe de nuevo a la pantalla de operaciones.	

Tabla 17. RF-17

REQUISITO FUNCIONAL (RF-14: Ficheros de mapas en entrenamiento)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
Si el entrenamiento ha concluido de manera correcta se tienen que mostrar por pantalla dos enlaces, uno con el mapa inicial y otro con el mapa entrenado. Ambos enlaces deben abrir un fichero de texto que debe estar alojado en un directorio temporal del servidor.	

Tabla 18. RF-18

REQUISITO FUNCIONAL (RF-15: Grafica evolución variable "J")	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
Si el entrenamiento ha concluido de manera correcta, se requiere mostrar un gráfico en la pantalla de entrenamiento que muestre la evolución de la variable "J", variable que nos muestra la suma total de las distancias de cada punto a la célula ganadora, por lo tanto el gráfico mostrará distancias por cada ciclo.	

Tabla 19. RF-19

REQUISITO FUNCIONAL (RF-16: Calibrado)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
Una vez pulsado el botón "Calibrar" de la pantalla de operaciones la aplicación nos muestra la pantalla de calibrado. Esta pantalla en primer lugar nos debe mostrar un seleccionable donde poder seleccionar un nuevo fichero para realizar el calibrado. Seleccionar este fichero es opcional, si el usuario lo dejará en blanco el calibrado se realizaría con el fichero de entrenamiento. Está pantalla debe incluir un botón "Volver" que al pulsarlo nos lleve de nuevo a la pantalla de operaciones. Además debe contener un botón "Calibrar" que al pulsarlo se invoque el algoritmo de calibrado y nos redirija a la pantalla de resultado del calibrado.	

Tabla 20. RF-20

REQUISITO FUNCIONAL (RF-17: Pantalla resultado calibrado)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
Una vez ejecutado el algoritmo de calibrado la aplicación nos llevará a la pantalla de resultado del calibrado. En esta pantalla se mostrará al usuario por pantalla si se ha producido algún tipo de error durante el calibrado. En caso contrario se debe mostrar un mensaje al usuario informándole que el algoritmo ha terminado de manera correcta. Se informará al usuario que en está pantalla puede volver a ejecutar el algoritmo de calibrado de nuevo. La pantalla debe incluir un seleccionable para que el usuario pueda seleccionar un nuevo fichero con el que volver a ejecutar el algoritmo de calibrado. Además debe contener un botón “Calibrar” que al pulsarlo se ejecute de nuevo el calibrado. De no haber seleccionado ningún nuevo fichero el algoritmo se debe ejecutar con el fichero de entrenamiento. Al ejecutarse de nuevo el calibrado en esta pantalla de nuevo nos mandará a esta pantalla de resultado de calibrado. La pantalla debe incluir un botón “Volver” que al pulsarlo nos lleve de nuevo a la pantalla de operaciones.	

Tabla 21. RF-21

REQUISITO FUNCIONAL (RF-18: Fichero de calibrado)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
En la pantalla de resultado de calibrado se debe mostrar un enlace que al pulsarlo abra un fichero de texto, que la aplicación previamente haya guardado en un directorio temporal del servidor, que muestre el resultado del mapa calibrado.	

Tabla 22. RF-22

REQUISITO FUNCIONAL (RF-19: Clasificado)

Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
Una vez pulsado el botón "Clasificar" de la pantalla de operaciones la aplicación nos muestra la pantalla de clasificado. Esta pantalla en primer lugar nos debe mostrar un seleccionable donde poder seleccionar un nuevo fichero para realizar el clasificado de los datos. Seleccionar este fichero es opcional, si el usuario lo dejará en blanco el calibrado se realizaría con el fichero de entrenamiento. Está pantalla debe incluir un botón "Volver" que al pulsarlo nos lleve de nuevo a la pantalla de operaciones. Además debe contener un botón "Clasificar" que al pulsarlo se invoque el algoritmo de clasificado y nos redirija a la pantalla de resultado del clasificado.	

Tabla 23. RF-23

REQUISITO FUNCIONAL (RF-20: Pantalla resultado clasificado)	
Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
Una vez ejecutado el algoritmo de clasificado la aplicación nos llevará a la pantalla de resultado del clasificado. En esta pantalla se mostrará al usuario por pantalla si se ha producido algún tipo de error durante el clasificado. En caso contrario se debe mostrar un mensaje al usuario informándole que el algoritmo ha terminado de manera correcta. Se informará al usuario que en está pantalla puede volver a ejecutar el algoritmo de clasificado de nuevo. La pantalla debe incluir un seleccionable para que el usuario pueda seleccionar un nuevo fichero con el que volver a ejecutar el algoritmo de clasificado. Además debe contener un botón "Clasificar" que al pulsarlo se ejecute de nuevo el clasificado. De no haber seleccionado ningún nuevo fichero el algoritmo se debe ejecutar con el fichero de entrenamiento. Al ejecutarse de nuevo el clasificado en esta pantalla de nuevo nos mandará a esta pantalla de resultado de clasificado. La pantalla debe incluir un botón "Volver" que al pulsarlo nos lleve de nuevo a la pantalla de operaciones.	

Tabla 24. RF-24

REQUISITO FUNCIONAL (RF-21: Fichero de clasificado)

Prioridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Fuente:** ☒ Cliente ☐ Desarrollador
Necesidad: ☒**Esencial** ☐ **Deseable** ☐ **Opcional**	
Claridad: ☒ **Alta** ☐ **Media** ☐ **Baja**	**Verificabilidad:**☒ Alta ☐ Media ☐ Baja
Estabilidad: Estable	
DESCRIPCION	
En la pantalla de resultado de clasificado se debe mostrar un enlace que al pulsarlo abra un fichero de texto, que la aplicación previamente haya guardado en un directorio temporal del servidor, que muestre el resultado con los datos clasificados.	

Tabla 25. RF-25

2 Casos de uso

A continuación se presentan los casos de uso de la aplicación en referencia a los requisitos de usuario.

CASOS DE USO (CU-01: Inicio práctica)	
Fuente:	RF-01
Actores:	Usuario
Objetivo:	Permite que el usuario comience la práctica.
Precondiciones:	
Postcondiciones:	La aplicación mostrará la pantalla de carga de datos.
Escenario básico	
El actor abre un navegador y ejecuta la url de la aplicación. El sistema presenta por pantalla la página de bienvenida de la aplicación con un menú horizontal. Una de las opciones del menú es la opción "Inicio Práctica". El usuario pincha esta opción del menú.	
Escenario alternativo	

Tabla 26: CU-01

CASOS DE USO (CU-02: Salir de la aplicación)	
Fuente:	RF-01
Actores:	Usuario
Objetivo:	Permite al usuario salir de la aplicación.
Precondiciones:	Estar en alguna pantalla de la aplicación.
Postcondiciones:	Se saldrá de la aplicación cerrando la sesión del usuario.
Escenario básico	
En cualquier pantalla de la aplicación, ya que el menú es común a todas las pantallas, el actor pinchará en la opción del menú "Salir".	
Escenario alternativo	

Tabla 27: CU-02

CASOS DE USO (CU-03: Cargar datos)	
Fuente:	RF-02, RF-03, RF-04
Actores:	Usuario
Objetivo:	Permite que el usuario cargue en memoria la ruta del fichero de

	entrenamiento y la ruta del fichero con un mapa inicial, ésta última sí fue seleccionada por el usuario.
Precondiciones:	Encontrarse en la pantalla de carga de datos.
Postcondiciones:	El fichero de entrenamiento se ha cargado en memoria y si se había seleccionado el fichero con el mapa inicial también se habrá cargado en memoria. El sistema redirige a la pantalla de carga de parámetros.
Escenario básico	
El sistema ha presentado la pantalla de carga de datos con dos seleccionables, uno para poder seleccionar el fichero de entrenamiento y otro para poder seleccionar el fichero con el mapa inicial. El usuario selecciona el fichero de entrenamiento y si quiere selecciona el fichero con el mapa inicial. A continuación el usuario pincha en el botón "Cargar Datos". El sistema carga en memoria las rutas de los ficheros seleccionados por el usuario y redirige a la pantalla de carga de parámetros.	
Escenario alternativo	
El usuario pulsa el botón "Cargar Datos" sin haber seleccionado el fichero de entrenamiento. El sistema muestra por pantalla al usuario un mensaje advirtiéndole que es obligatorio seleccionar el fichero de entrenamiento.	

Tabla 28: CU-03

CASOS DE USO (CU-04: Cargar parámetros)	
Fuente:	RF-06, RF-07, RF-08, RF-09, RF-10, RF-011
Actores:	Usuario
Objetivo:	Permite que el usuario cargue en memoria los parámetros de entrada para el algoritmo de entrenamiento.
Precondiciones:	Encontrarse en la pantalla de carga de parámetros.
Postcondiciones:	Se han cargado en memoria los parámetros de entrada del algoritmo de entrenamiento. El sistema redirige a la pantalla de carga de parámetros.
Escenario básico	
El sistema presenta la pantalla de carga de parámetros. Esta pantalla contendrá los siguientes campos: - Dimensión X del mapa: Será un campo de texto que reciba un número entero y será obligatorio. - Dimensión Y del mapa: Será un campo de texto que reciba un número entero y será obligatorio. - Número de ciclos: Será un campo de texto que reciba un número entero y será obligatorio. - Tasa de aprendizaje: Será un campo de texto que reciba un número con posibles decimales y será obligatorio. - Variación de la tasa de aprendizaje: Este campo será un seleccionable donde el	

usuario pueda elegir una de los siguientes tipos de tasas de aprendizaje: lineal, logarítmica o sin variación. - Límite de vecindario: Será un campo de texto que reciba un número entero y será obligatorio. El actor debe rellenar todos los datos y a continuación pinchar en el botón "Cargar Datos". El sistema cargará en memoria todos los parámetros de entrada para que posteriormente puedan ser utilizados y a continuación redirigirá a la pantalla de operaciones.
Escenario alternativo
El usuario pincha en el botón "Cargar Datos" sin haber rellenado algunos de los parámetros de entrada. El sistema muestra por pantalla un mensaje al usuario advirtiéndole que debe rellenar todos los campos del formulario antes de avanzar a la pantalla de operaciones y permanece en la pantalla de carga de parámetros.

Tabla 29: CU-04

CASOS DE USO (CU-05: Limpiar formularios)	
Fuente:	RF-05
Actores:	Usuario
Objetivo:	Permite que el usuario limpie los formularios de datos de la aplicación.
Precondiciones:	Encontrarse en una página que pida datos de entrada al usuario.
Postcondiciones:	Los campos del formulario quedarán vacíos.
Escenario básico	
El actor debe encontrarse en una pantalla que presente algún campo que pueda ser rellenado por el usuario. El actor pinchará en el botón "Limpiar". El sistema inicializa todos los campos del formulario dejándolos vacíos.	
Escenario alternativo	

Tabla 30: CU-05

CASOS DE USO (CU-06: Operar)	
Fuente:	RF-12
Actores:	Usuario
Objetivo:	Permite que el usuario pueda realizar alguna de las siguientes operaciones: - Entrenar. - Calibrar.

	- Clasificar
Precondiciones:	Encontrarse en la pantalla de operaciones.
Postcondiciones:	El usuario realizará la operación que haya elegido.
Escenario básico	
El sistema presenta la pantalla de operaciones. Esta pantalla tiene que tener tres botones, "Entrenar", "Calibrar" y "Clasificar" que invoquen a las tres operaciones que se pueden realizar. El usuario realiza una de las tres siguientes acciones: 1) El usuario pincha en el botón "Entrenar". El sistema comienza el algoritmo de entrenamiento y redirige a la pantalla de entrenamiento. 2) El usuario pincha en el botón "Clasificar" El sistema envía a la pantalla de calibrado. 3) El usuario pincha en el botón "Clasificar". El sistema redirige a la pantalla de clasificado.	
Escenario alternativo	
4) El usuario pincha en el botón "Calibrar" sin antes haber realizado el entrenamiento del mapa. El sistema muestra por pantalla un mensaje al usuario diciéndole que antes de realizar el calibrado hay que entrenar el mapa. 5) El usuario pincha en el botón "Clasificar" sin antes haber calibrado el mapa. El sistema muestra un mensaje por pantalla al usuario advirtiéndole que antes de realizar el clasificado de los datos tiene que haber realizado el calibrado del mapa.	

Tabla 31: CU-06

CASOS DE USO (CU-07: Entrenar)	
Fuente:	RF-13, RF-15
Actores:	Usuario
Objetivo:	Permite al usuario realizar el entrenamiento del mapa.
Precondiciones:	El usuario tiene que haber pulsado el botón "Entrenar".
Postcondiciones:	Se realiza el entrenamiento del mapa.
Escenario básico	
El usuario ha pulsado el botón "Entrenar" El sistema comienza el algoritmo de entrenamiento a partir de los datos y parámetros recogidos anteriormente. Una vez acabado el algoritmo el sistema redirige a la pantalla de entrenamiento que mostrará al usuario un mensaje informándole que el algoritmo ha concluido de manera correcta, Además el sistema mostrará por pantalla dos enlaces: - Un enlace al fichero del mapa inicial - Otro enlace al fichero del mapa entrenado. El sistema también debe mostrar un gráfico al usuario que represente la evolución de la variable "J" donde se pueda observar que el algoritmo se ha ejecutado de manera correcta observando como en cada ciclo la variable va disminuyendo.	

Además la pantalla debe contener el botón “Aceptar” que cuando el usuario pulsa dicho botón la aplicación nos redirige de nuevo a la pantalla de operaciones.
Escenario alternativo
Si se ha producido algún tipo de error durante el algoritmo de entrenamiento, el sistema mostrará un mensaje por pantalla al usuario informándole del error que se ha producido.

Tabla 32: CU-07

CASOS DE USO (CU-08: Visualizar fichero mapa entrenado y mapa inicial)	
Fuente:	RF-014
Actores:	Usuario
Objetivo:	Permite que el usuario visualice el fichero con el mapa inicial y el fichero con el mapa entrenado.
Precondiciones:	Se tiene que haber realizado el entrenamiento correctamente.
Postcondiciones:	Visualización de los ficheros.
Escenario básico	
Una vez realizado el entrenamiento el sistema presenta la pantalla de entrenamiento donde aparecen dos enlaces, uno al fichero del mapa inicial y otro al fichero del mapa entrenado. El usuario pincha en el enlace del mapa inicial. El sistema abre un fichero de texto donde se puede observar el mapa inicial. El usuario pincha en el enlace del mapa entrenado. El sistema abre un fichero de texto con el mapa entrenado.	
Escenario alternativo	

Tabla 33: CU-08

CASOS DE USO (CU-09: Calibrar)	
Fuente:	RF-16, RF-17
Actores:	Usuario
Objetivo:	Permite que el usuario realice el calibrado del mapa.
Precondiciones:	Haber pinchado en la pantalla de operaciones el botón “Calibrar”.
Postcondiciones:	Una vez se haya ejecutado el algoritmo correctamente obtenemos el mapa calibrado.
Escenario básico	

El usuario pincha en botón “Calibrar” de la pantalla de operaciones.
El sistema presenta la pantalla de calibrado donde podemos elegir un nuevo fichero de datos para realizar el calibrado.

1) El usuario pincha en el botón “Calibrar” sin haber seleccionado ningún fichero de nuevo.
   El sistema realiza el calibrado con los datos del fichero de entrenamiento y redirige a la pantalla de resultado del calibrado.
   Esta pantalla muestra al usuario un mensaje informándole que el algoritmo ha concluido correctamente y le da la opción de si quiere volver a realizar otra vez el calibrado puede hacerlo. Puede volver a realizar el calibrado con el fichero de entrenamiento o volviendo a seleccionar otro fichero.
   El sistema presenta en esta pantalla también un enlace a un fichero de texto con el mapa calibrado.

Escenario alternativo

2) El usuario pincha el botón “Volver”.
   El sistema redirige de nuevo a la pantalla de operaciones
3) El usuario selecciona un nuevo fichero de datos para realizar el calibrado y a continuación pincha en el botón “Calibrar”.
   El sistema realiza el calibrado del mapa a partir del nuevo fichero de datos seleccionado, redirige a la pantalla de resultado de calibrado y continua de igual manera que en el punto 1.
4) El usuario realiza el calibrado y se produce un error.
   El sistema muestra por pantalla al usuario un mensaje informándole del error que se ha producido.

Tabla 34: CU-09

CASOS DE USO (CU-10: Visualizar fichero mapa calibrado)	
Fuente:	RF-18
Actores:	Usuario
Objetivo:	Permite que el usuario visualice el fichero con el mapa calibrado.
Precondiciones:	Se tiene que haber realizado el calibrado correctamente.
Postcondiciones:	El usuario visualiza el fichero con el mapa calibrado.

Escenario básico

Una vez realizado el calibrado el sistema presenta la pantalla de resultado de calibrado donde aparece un enlace al fichero calibrado.
El usuario pincha en el enlace del mapa calibrado.
El sistema abre un fichero de texto donde se puede observar el mapa calibrado.

Escenario alternativo

Tabla 35: CU-10

CASOS DE USO (CU-11: Clasificar)	
Fuente:	RF-19, RF-20
Actores:	Usuario
Objetivo:	Permite que el usuario realice el clasificado de los datos del fichero que haya elegido.
Precondiciones:	Haber pinchado en la pantalla de operaciones el botón "Clasificar".
Postcondiciones:	Una vez se haya ejecutado el algoritmo correctamente obtenemos los datos clasificados.

Escenario básico

El usuario pincha en botón "Clasificar" de la pantalla de operaciones.
El sistema presenta la pantalla de clasificado donde podemos elegir un nuevo fichero de datos para realizar el clasificado.

1) El usuario pincha en el botón "Clasificar" sin haber seleccionado ningún fichero de nuevo.
   El sistema realiza el clasificado de los datos del fichero de entrenamiento y redirige a la pantalla de resultado del clasificado.
   Esta pantalla muestra al usuario un mensaje informándole que el algoritmo ha concluido correctamente y le da la opción de si quiere volver a realizar otra vez el clasificado puede hacerlo. Puede volver a realizar el clasificado con el fichero de entrenamiento o volviendo a seleccionar otro fichero.
   El sistema presenta en esta pantalla también un enlace a un fichero de texto con el mapa clasificado.

Escenario alternativo

1) El usuario pincha el botón "Volver".
   El sistema redirige de nuevo a la pantalla de operaciones
2) El usuario selecciona un nuevo fichero de datos para realizar el clasificado y a continuación pincha en el botón "Clasificar".
   El sistema realiza el clasificado a partir del nuevo fichero de datos seleccionado, redirige a la pantalla de resultado de clasificado y continua de igual manera que en el punto 1.
3) El usuario realiza el clasificado y se produce un error.
   El sistema muestra por pantalla al usuario un mensaje informándole del error que se ha producido.

Tabla 36: CU-11

CASOS DE USO (CU-12: Visualizar fichero datos clasificados)	
Fuente:	RF-21
Actores:	Usuario
Objetivo:	Permite que el usuario visualice el fichero con los datos clasificados.
Precondiciones:	Se tiene que haber realizado el clasificado correctamente.
Postcondiciones:	El usuario visualiza el fichero con los datos clasificados.

Escenario básico

Una vez realizado el clasificado el sistema presenta la pantalla de resultado de clasificado donde aparece un enlace al fichero con los datos clasificados.

El usuario pincha en el enlace del fichero clasificado. El sistema abre un fichero de texto donde se pueden observar los datos clasificados.
Escenario alternativo

Tabla 37: CU-12

3 Plan de pruebas

A continuación se presenta el plan de pruebas a seguir para comprobar el buen funcionamiento de la aplicación cumpliendo con los requisitos funcionales que se definieron antes de desarrollar la aplicación.

PRUEBA 1 (PU-01: Inicio aplicación)	
Funcionalidad	Inicio aplicación.
Descripción	Mediante esta prueba comprobaremos si el inicio a la práctica se realiza de manera correcta.
Caso de uso	CU-01
Pasos	1) Abrir un navegador y ejecutar la url de la aplicación. 2) El usuario pincha en la opción del menú "Inicio Práctica".
Resultado esperado	1) La aplicación muestra la página de bienvenida con un menú horizontal donde una de las opciones es "Inicio Práctica". 2) La aplicación mostrará la pantalla de carga de datos.

Tabla 38: PU-01

PRUEBA 2 (PU-02: Salir de la práctica)	
Funcionalidad	Salir de la aplicación.
Descripción	Mediante esta prueba comprobaremos como el usuario puede salir de la aplicación.
Caso de uso	CU-02
Pasos	1) Para realizar esta prueba nos debemos encontrar en alguna pantalla de la aplicación ya que todas contienen el menú horizontal. Pincharemos en la opción del menú "Inicio Práctica".
Resultado esperado	1) Comprobamos que hemos salido de la aplicación cerrando la sesión del usuario

Tabla 39: PU-02

PRUEBA 3 (PU-03: Cargar datos)	
Funcionalidad	Cargar datos.
Descripción	Mediante esta prueba comprobaremos que se cargan en memoria de manera correcta las rutas de los ficheros de entrenamiento y del mapa inicial.
Caso de uso	CU-03
Pasos	1) Pinchar en el menú la opción "Inicio Práctica". 2) Seleccionar un fichero de entrenamiento y un fichero con el

	mapa inicial y pulsar el botón “Cargar Datos”. 3) Pulsar el botón “Cargar Datos” sin haber seleccionado el fichero de entrenamiento. 4) Pulsar el botón “Cargar Datos” sin haber seleccionado el fichero con el mapa inicial y si el fichero de entrenamiento.
Resultado esperado	1) Comprobar que se carga la pantalla de carga de datos con dos seleccionables, uno para poder seleccionar el fichero de entrenamiento y otro para poder seleccionar el fichero con el mapa inicial. 2) Comprobar que las rutas de los fichero se cargan en memoria y que el sistema redirige a la pantalla de carga de parámetros. 3) Comprobar que la aplicación muestra un mensaje por pantalla al usuario advirtiéndole que debe seleccionar el fichero de entrenamiento obligatoriamente, quedándose en la misma pantalla. 4) Comprobar que se pasa a la siguiente pantalla ya que el fichero con el mapa inicial es opcional.

Tabla 40: PU-03

PRUEBA 4 (PU-04: Cargar parámetros)	
Funcionalidad	Cargar parámetros.
Descripción	Mediante esta prueba comprobaremos que se cargan en memoria de manera correcta los parámetros de la aplicación.
Caso de uso	CU-04
Pasos	1) Pasamos de la pantalla de carga de datos a la pantalla de carga de parámetros. 2) Rellenamos todos los campos del formulario y pulsamos el botón “Cargar Datos”. 3) Pulsamos en el botón “Cargar Datos” sin haber rellenado alguno de los campos del formulario.
Resultado esperado	1) Comprobamos que la pantalla de carga de parámetros muestra los siguientes campos: - Dimensión X del mapa. - Dimensión Y del mapa. - Número de ciclos. - Tasa de aprendizaje. - Variación de la tasa de aprendizaje. - Límite de vecindario. 2) Comprobar que se cargan correctamente en memoria los parámetros y que la aplicación redirige a la pantalla de operaciones. 3) Comprobamos como la aplicación muestra un mensaje por pantalla al usuario informándole que tiene que rellenar el campo que haya dejado en blanco.

Tabla 41: PU-04

PRUEBA 5 (PU-05: Limpiar formularios)	
Funcionalidad	Limpiar formularios.
Descripción	Comprobaremos que se limpian correctamente los formularios de datos de la aplicación..
Caso de uso	CU-05
Pasos	1) Entramos en la pantalla de carga de datos y a continuación rellenamos todos los campos del formulario y pinchamos en el botón "Limpiar". 2) Realizamos el mismo proceso en la pantalla de carga de parámetros. 3) En la pantalla de calibrado seleccionamos un fichero y pinchamos el botón "Limpiar". 4) Realizamos la misma prueba en la pantalla de clasificado.
Resultado esperado	1) Comprobamos que todos los campos del formulario se han inicializado correctamente. 2) Comprobamos que todos los campos del formulario se han inicializado correctamente. 3) Comprobamos que el campo se ha inicializado. 4) Comprobamos que el campo se ha inicializado.

Tabla 42: PU-05

PRUEBA 6 (PU-06:Operar)	
Funcionalidad	Operar.
Descripción	Mediante esta prueba comprobaremos sí la pantalla de operaciones funciona de manera correcta.
Caso de uso	CU-06
Pasos	1) Pulsamos el botón "Cargar datos" en la pantalla de carga de parámetros y pasamos a la pantalla de operaciones. 2) Pinchamos el botón "Entrenar". 3) Pinchamos el botón "Calibrar" después de haber realizado el entrenamiento. 4) Pinchamos el botón "Clasificar" después de haber realizado el calibrado. 5) Pinchamos el botón "Calibrar" sin haber realizado el entrenamiento. 6) Pinchamos el botón "Calibrar" sin haber realizado el calibrado.
Resultado esperado	1) Comprobamos que la pantalla de operaciones contiene cuatro botones: - "Entrenar" - "Calibrar" - "Clasificar" - "Limpiar" 2) Comprobamos que la aplicación ejecuta el algoritmo de entrenamiento.

	3) Comprobamos que la aplicación nos redirige a la pantalla de calibrado. 4) Comprobamos que la aplicación nos redirige a la pantalla de clasificado. 5) Comprobamos que si el usuario no ha realizado el entrenamiento antes de realizar el calibrado del mapa el sistema muestra por pantalla un mensaje al usuario informándole que antes de realizar el calibrado hay que haber entrenado el mapa. 6) Comprobamos que si el usuario no ha realizado el calibrado del mapa antes de realizar el clasificado de los datos el sistema muestra por pantalla un mensaje al usuario informándole que antes de realizar el clasificado hay que haber realizado el calibrado del mapa.

Tabla 43: PU-06

PRUEBA 7 (PU-07: Entrenar)	
Funcionalidad	Entrenar.
Descripción	Mediante esta prueba comprobaremos si el algoritmo de entrenamiento funciona correctamente.
Caso de uso	CU-07
Pasos	1) Pinchamos en el botón "Entrenar" en la pantalla de operaciones. 2) Realizamos de nuevo un entrenamiento forzando que ocurra un error durante la ejecución del algoritmo. 3) Pinchamos en el botón "Aceptar" de la pantalla de resultado de entrenamiento.
Resultado esperado	1) Comprobamos que se ejecuta el algoritmo de entrenamiento correctamente y una vez terminado la aplicación nos redirige a la pantalla de resultado de entrenamiento. Esta pantalla debe mostrar los siguientes elementos: - Un mensaje por pantalla al usuario informándole que el algoritmo de entrenamiento se ha ejecutado de manera correcta. - Un botón "Aceptar". - Un enlace a un fichero de texto, que ha sido guardado en el servidor, que contiene el mapa inicial utilizado durante el algoritmo. - Otro enlace a un fichero de texto que contiene el mapa entrenado. - También debe de mostrar un gráfico donde se muestre la evolución de la variable "J" por cada ciclo del algoritmo de entrenamiento donde debemos observar la tendencia a disminuir en cada ciclo de esta variable, ya que la distancia de las células a los puntos en cada ciclo debe disminuir al ejecutarse el algoritmo correctamente. Si esta variable no

	disminuyera es que el algoritmo no se ha ejecutado de manera correcta 2) Comprobamos que la aplicación nos redirige a la pantalla de resultado del entrenamiento y muestra por pantalla al usuario un mensaje informándole del error que se ha producido durante la ejecución. 3) Comprobamos que la aplicación nos redirige de nuevo a la pantalla de operaciones.

Tabla 44: PU-07

PRUEBA 8 (PU-08: Visualizar fichero mapa entrenado y mapa inicial)	
Funcionalidad	Visualizar fichero mapa entrenado y mapa inicial.
Descripción	Por medio de esta prueba comprobaremos si el fichero con el mapa entrenado se visualiza correctamente así como el fichero con el mapa inicial.
Caso de uso	CU-08
Pasos	1) Pinchamos en el enlace del fichero del mapa inicial de la pantalla de entrenado. 2) Punchamos en el enlace del fichero de entrenamiento.
Resultado esperado	1) Comprobamos que el fichero de texto del mapa inicial se visualiza correctamente. También comprobamos el formato del fichero que deberá tener una fila por cada célula y en cada célula las coordenadas de ésta en el mapa. 2) Comprobamos que el fichero de texto del mapa entrenado se visualiza correctamente. Además comprobaremos el formato del fichero de entrenamiento que será el mismo que el del mapa inicial.

Tabla 45: PU-08

PRUEBA 9 (PU-09: Calibrar)	
Funcionalidad	Calibrar.
Descripción	Mediante esta prueba comprobaremos si el algoritmo de calibrado se ejecuta correctamente.
Caso de uso	CU-09
Pasos	1) Pinchamos en el botón “Calibrar” en la pantalla de operaciones y vamos a la pantalla de calibrado. 2) Pinchar en el botón “Calibrar” sin haber seleccionado un nuevo fichero. 3) Seleccionamos un nuevo fichero desde el seleccionable y pinchamos en el botón “Calibrar”. 4) Pinchamos en el botón “Volver”. 5) Pinchamos en el botón “Calibrar” y forzamos a que ocurra un error.

Resultado esperado	1) Comprobamos que la pantalla de calibrado contiene un seleccionable para poder realizar el calibrado con un nuevo fichero. Además debe tener tres botones: "Calibrar", "Limpiar", y "Volver". 2) Comprobamos que se realiza el calibrado de datos a partir del fichero de entrenamiento y a continuación el sistema redirige a la pantalla de resultado de calibrado donde se debe mostrar un mensaje al usuario informándole que el calibrado se ha ejecutado correctamente y un enlace a un fichero de texto que contiene el mapa calibrado. La pantalla presenta de nuevo los mismos campos que la pantalla de calibrado para poder realizar otro calibrado de datos a partir de un nuevo fichero o del de entrenamiento. 3) Comprobamos que se realiza el calibrado del mapa a partir del nuevo fichero seleccionado y la aplicación redirige a la pantalla de resultado de calibrado al igual que en el punto anterior. 4) Comprobamos que la aplicación nos redirige de nuevo a la pantalla de operaciones. 5) Comprobamos que la aplicación muestra por pantalla al usuario el error que se ha producido.

Tabla 46: PU-09

PRUEBA 10 (PU-10: Visualizar fichero mapa calibrado)	
Funcionalidad	Visualizar fichero mapa calibrado.
Descripción	Por medio de esta prueba comprobaremos si el fichero con el mapa calibrado se visualiza correctamente.
Caso de uso	CU-10
Pasos	1) Pinchamos en el enlace del fichero del mapa calibrado de la pantalla de resultado de calibrado.
Resultado esperado	1) Comprobamos que el fichero de texto del mapa calibrado se visualiza correctamente. Además comprobaremos el formato del fichero del mapa calibrado que será el mismo que el del mapa inicial.

Tabla 47: PU-10

PRUEBA 11 (PU-11: Clasificar)	
Funcionalidad	Clasificar.
Descripción	Mediante esta prueba comprobaremos si el algoritmo de clasificado funciona correctamente.
Caso de uso	CU-11
Pasos	1) Pinchamos en el botón "Clasificar" en la pantalla de operaciones y vamos a la pantalla de clasificado.

	2) Pinchar en el botón “Clasificar” sin haber seleccionado un nuevo fichero. 3) Seleccionamos un nuevo fichero desde el seleccionable y pinchamos en el botón “Clasificar”. 4) Pinchamos en el botón “Volver”. 5) Pinchamos en el botón “Clasificar” y forzamos a que ocurra un error.
Resultado esperado	1) Comprobamos que la pantalla de clasificado contiene un seleccionable para poder realizar el clasificado con un nuevo fichero. Además debe tener tres botones: “Clasificar”, “Limpiar”, y “Volver”. 2) Comprobamos que se realiza el clasificado de datos a partir del fichero de entrenamiento y a continuación el sistema redirige a la pantalla de resultado de clasificado donde se debe mostrar un mensaje al usuario informándole que el clasificado se ha ejecutado correctamente y un enlace a un fichero de texto que contiene el fichero clasificado. La pantalla presenta de nuevo los mismos campos que la pantalla de clasificado para poder realizar otro clasificado de datos a partir de un nuevo fichero o del de entrenamiento. 3) Comprobamos que se realiza el clasificado del mapa a partir del nuevo fichero seleccionado y la aplicación redirige a la pantalla de resultado de clasificado al igual que en el punto anterior. 4) Comprobamos que la aplicación nos redirige de nuevo a la pantalla de operaciones. 5) Comprobamos que la aplicación muestra por pantalla al usuario el error que se ha producido.

Tabla 48: PU-11

PRUEBA 12 (PU-12: Visualizar fichero datos clasificados)	
Funcionalidad	Visualizar fichero datos clasificados.
Descripción	Por medio de esta prueba comprobaremos si el fichero con los datos clasificados se visualiza correctamente.
Caso de uso	CU-12
Pasos	1) Pinchamos en el enlace del fichero de los datos clasificados de la pantalla de resultado de clasificado.
Resultado esperado	1) Comprobamos que el fichero de texto de los datos clasificados se visualiza correctamente. Además comprobaremos el formato del fichero de los datos clasificados que deberá tener en cada fila un dato y por cada dato la célula a la que pertenece y si ésta tenía etiqueta también vendrá al final de la fila dicha etiqueta.

Tabla 49: PU-12